USER GUIDE

AWS User Guide

DuploCloud

TABLE OF CONTENTS

Introduction

DuploCloud is a rules-based orchestration engine. It is the subject matter expert in digital form. It is not a DevOps tool that requires a cloud expert to operate, but instead it is the cloud expert itself. The software has a resume with over 500 skills that include AWS, Azure, Kubernetes, Security and IoT among others. It allows users to build applications via a simple declarative interface and not have to deal with low level infrastructure details like VPC, security groups, IAM, Jenkins setup, etc. These constructs are still in play, but the DuploCloud software abstracts it away for you by auto-generating the configuration based on application needs.

DuploCloud is single tenant software that installs in your cloud account. Users interface with the software via the browser UI and/or API calls and all data and configuration stays within your cloud account. All configurations that have been created and applied by the software are transparently available to be reviewed and edited in your cloud account. Essentially, the software auto-generates the same automation scripts that a human being would have written manually. Here is an explainer video about the product:

DuploCloud Product Demo (AWS).mp4

Pre-requisites

- **Sign-in Account:** DuploCloud works off Google and O365 Oauth. To login to DuploCloud you either need a Google or Office365 account.
- **Docker Knowledge:** If you are deploying a Docker based microservice using DuploCloud then it is assumed that you are familiar with Docker.
    - You should have a docker image for your application that has been tested locally in your computer. Make sure it runs in detached mode (i.e., docker run -d option).
    - The Image should have been pushed to your repository. If the repository is private, then you will have to set the credentials for Docker hub in your DuploCloud account.
- **AWS SDK Familiarity:** If you are using AWS services like S3, DynamoDB, SQS, and SNS you must have a basic knowledge of how these services can be consumed. The interface to create these services via DuploCloud will be very declarative and self-explanatory. You do not need any access keys in your code to access these services. Use the AWS constructor that does not take credentials, but takes only region which must be US-West-2 unless otherwise specified by your enterprise administrator or during signup.

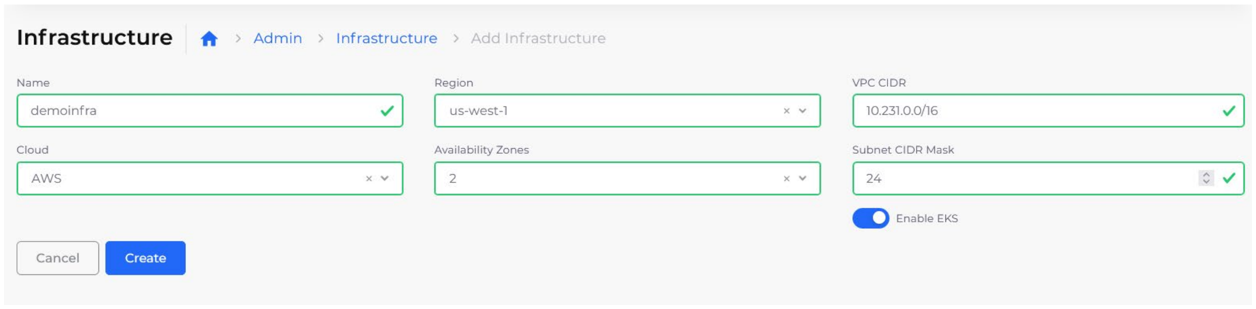
Terminologies

- **Infrastructure:** An infrastructure maps 1:1 with a VPC/VNET and can be in any region. Each infrastructure has a set of subnets spread across multiple availability zones. In AWS there is a NAT gateway for private subnets.
- **Tenant or Project:** Tenant or Project is a sandbox or unit of deployment. All resources in a given tenant are isolated from another tenant via Security groups and IAM policies (optionally VPC). For applications to be reachable outside the sandbox a Port mapping or ELB must be created.
- **User:** This is an individual with a user ID. Each user could have access to one more tenants/projects.
- **Host:** This is an EC2 instance or VM. This is where your application will run.
- **Service:** Service is your application code packaged as a single docker image and running as a set of one or more containers. It is specified as - image-name; replicas; env-variables; vol-mappings if any. DuploCloud also allows running applications that are not packaged as Docker images.
- **ELB:** A Service can be exposed outside of the tenant\project via an ELB and DNS name. ELB is defined as - Service name + container-port + External port + Internal-or-internet facing. Optionally, a wild card certificate can be chosen for SSL termination. You can choose to make it internal which will expose it only within your VPC to other applications.
- **DNS Name:** By default, when a Service is exposed via an ELB, DuploCloud will create a friendly DNS Name. A user can choose to edit this name. The domain name must have been configured in the system by the admin.
- **Docker Host or Fleet Host:** If a host is marked as part of the fleet, then DuploCloud will use it to deploy containers. If the user needs a host for development purposes, say a test machine, then they would mark it as not part of the pool or fleet.

AWS Guide

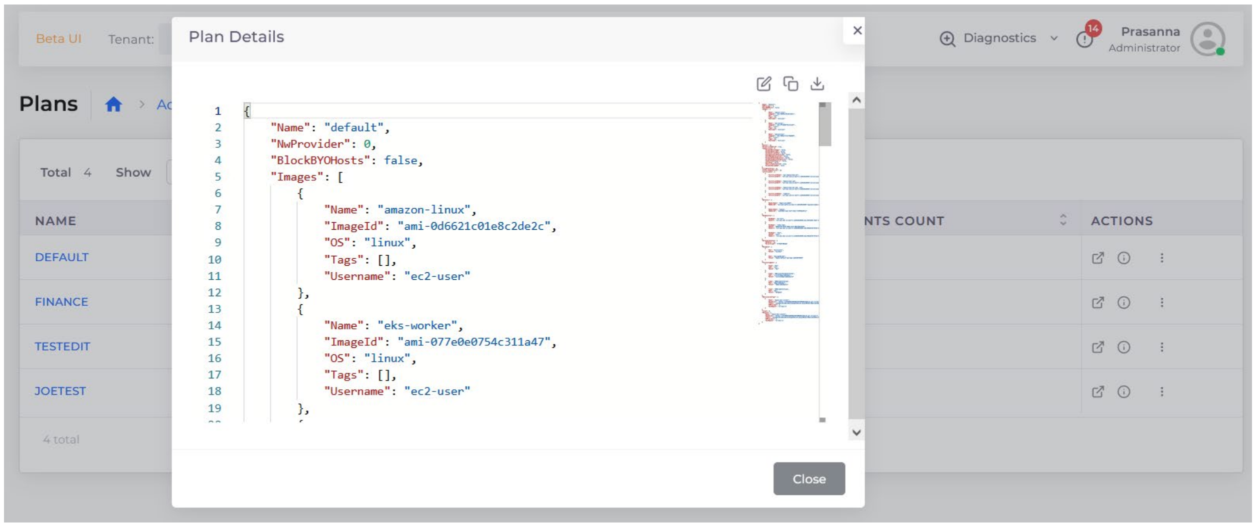
Infrastructure

A completely new isolated environment can be created for use cases from DuploCloud. Proceed to Administrator --> Infrastructure. Click on +Add button. It opens a form where it asks to enter basic information about the new environment.



Name: demoinfra	Region: us-west-1	VPC CIDR: 10.231.0.0/16
Cloud: AWS	Availability Zones: 2	Subnet CIDR Mask: 24
		Enable EKS

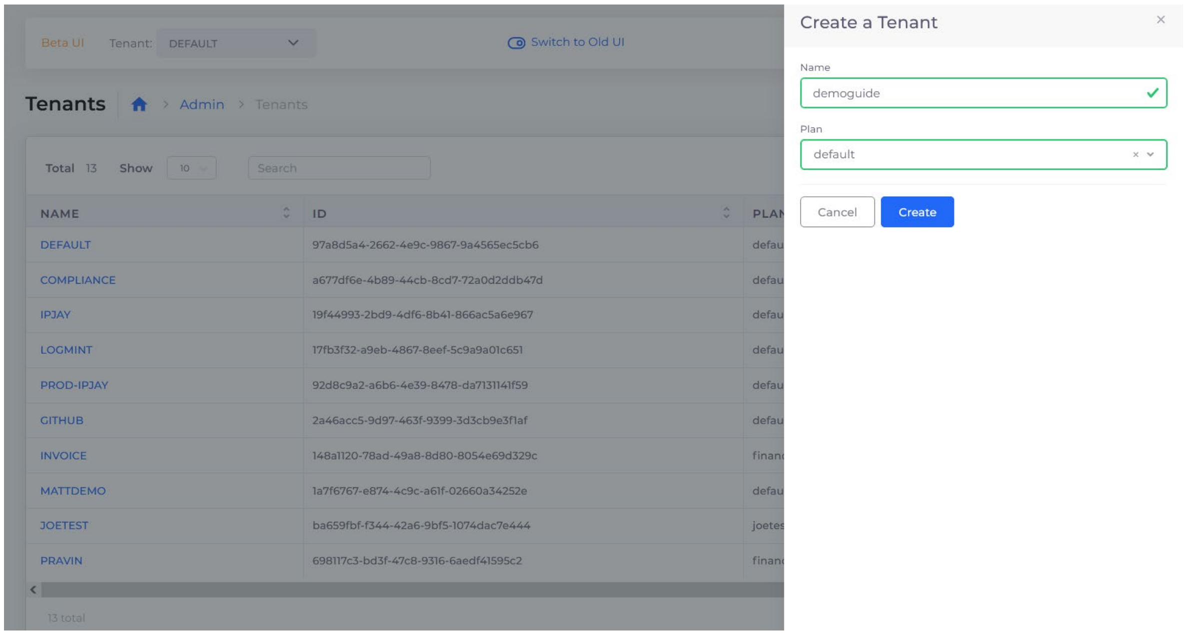
Once you submit the form, DuploCloud creates a new VPC with 2 subnets (private, public) in each availability zone, necessary security groups, NAT Gateway, Internet Gateway, Route tables, etc. DuploCloud also configures the VPC peering with the master VPC which is initially configured in DuploCloud. Infrastructure creation takes around 10mins to complete. Once its complete, a new plan with the same infrastructure name is automatically created and populated with the details of the newly created infrastructure. This plan can then be used to create the tenants.



```
{
    "Name": "default",
    "NwProvider": 0,
    "BlockBYOHosts": false,
    "Images": [
        {
            "Name": "amazon-linux",
            "ImageId": "ami-0d6621c01e8c2de2c",
            "OS": "linux",
            "Tags": [],
            "Username": "ec2-user"
        },
        {
            "Name": "eks-worker",
            "ImageId": "ami-077e0e0754c311a47",
            "OS": "linux",
            "Tags": [],
            "Username": "ec2-user"
        },
```

Tenant

Create a new Tenant from the already created plan. Give a min for the DuploCloud to create the Tenant specific SG and Instance profiles which acts as a security boundary.



Total 13 Show 10

NAME	ID	PLAN
DEFAULT	97a8d5a4-2662-4e9c-9867-9a4565ec5cb6	defau
COMPLIANCE	a677df6e-4b89-44cb-8cd7-72a0d2ddb47d	defau
IPJAY	19f44993-2bd9-4df6-8b41-866ac5a6e967	defau
LOGMINT	17fb3f32-a9eb-4867-8eef-5c9a9a01c651	defau
PROD-IPJAY	92d8c9a2-a6b6-4e39-8478-da7131141f59	defau
GITHUB	2a46acc5-9d97-463f-9399-3d3cb9e3f1af	defau
INVOICE	148a1120-78ad-49a8-8d80-8054e69d329c	finan
MATTDEMO	1a7f6767-e874-4c9c-a61f-02660a34252e	defau
JOETEST	ba659fbf-f344-42a6-9bf5-1074dac7e444	joetes
PRAVIN	698117c3-bd3f-47c8-9316-6aedf41595c2	finan

13 total

Create a Tenant

Name: demoguide

Plan: default

Quick Start – A Docker Microservice

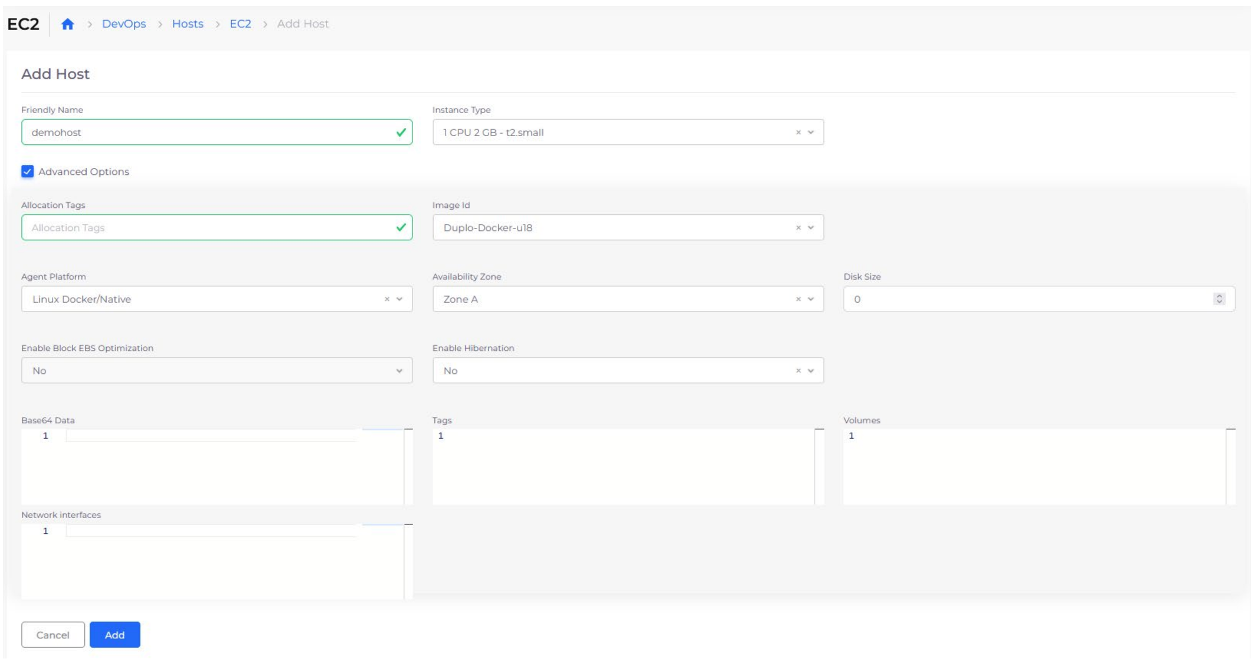
Deployment is a three-step process.

1. Create a Host (VM)
2. Deploy the service (Application)
3. Expose the application using LB

Watch this short video that shows how to package your code in a Docker image and deploy:

DuploCloud Product Demo (AWS).mp4

- **Create a Host** from the menu **DevOps → Hosts → EC2 → +Add** button above the table. Choose the desired instance type. The available instance types are set by your administrator or the plan you choose in DuploCloud. If you are not using this host for hosting containers, then set the pool as none. If you want a public IP for the host, then select the public subnet in the list of availability zones.


EC2 › DevOps › Hosts › EC2 › Add Host

Add Host

Friendly Name: demohost

Instance Type: 1 CPU 2 GB - t2.small

Advanced Options [✓]

Allocation Tags: Allocation Tags

Image Id: Duplo-Docker-u18

Agent Platform: Linux Docker/Native

Availability Zone: Zone A

Disk Size: 0

Enable Block EBS Optimization: No

Enable Hibernation: No

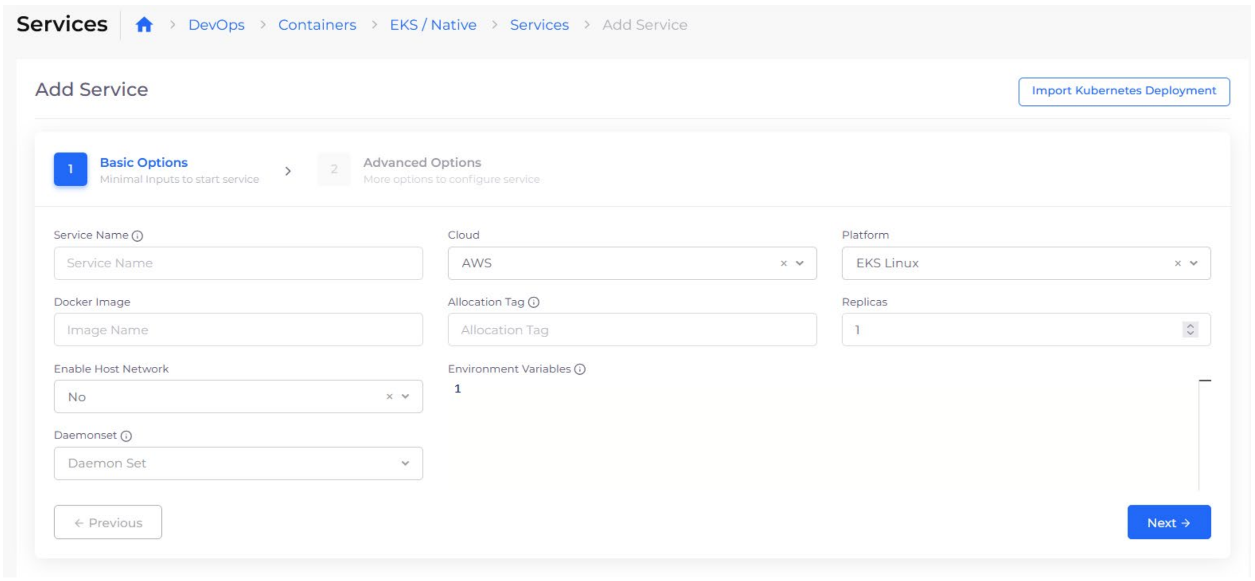
Base64 Data: 1

Tags: 1

Volumes: 1

Network interfaces: 1

- **Deploy a Service (application)** from the menu **DevOps → Containers → EKS/ Native → +Add** sign above the table. Give a name for your services (no spaces); number of replicas; Docker image; volumes (if any). The number of replicas must be less than or equal to the number of hosts in the fleet.


Services › DevOps › Containers › EKS / Native › Services › Add Service

Add Service

1 Basic Options — Minimal Inputs to start service

2 Advanced Options — More options to configure service

Service Name: Service Name

Cloud: AWS

Platform: EKS Linux

Docker Image: Image Name

Allocation Tag: Allocation Tag

Replicas: 1

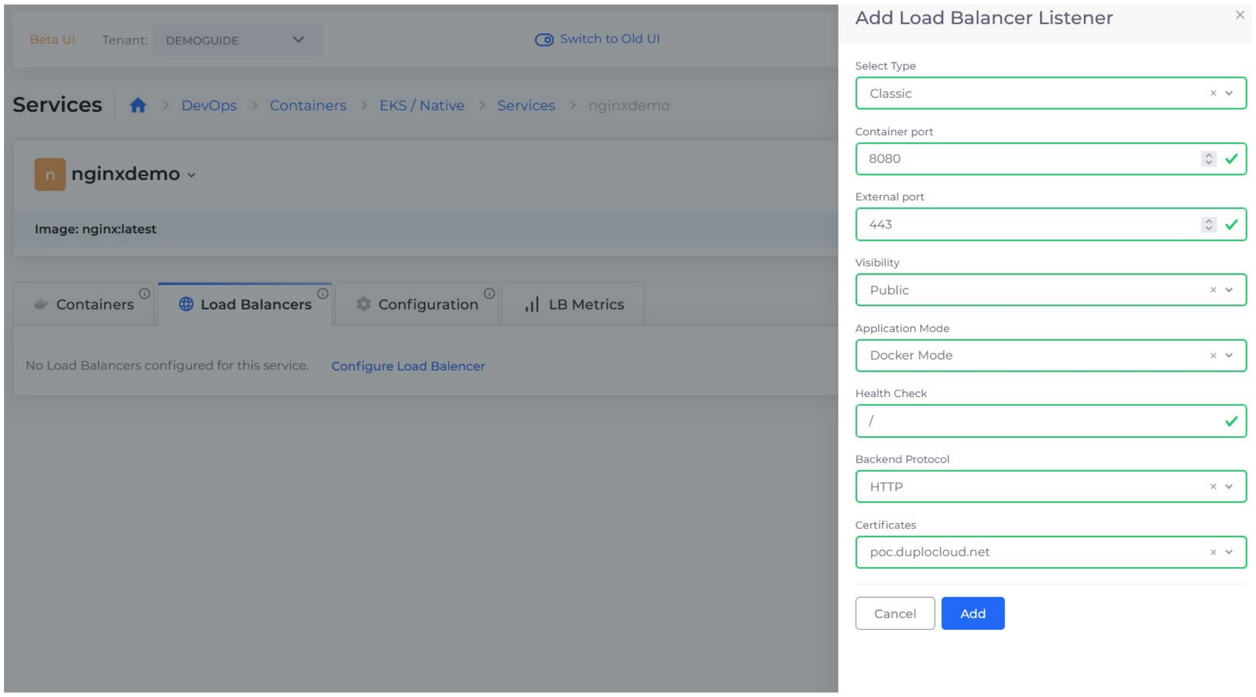
Enable Host Network: No

Environment Variables: 1

Daemonset: Daemon Set

- Create an ELB from the menu **DevOps → Containers → EKS/ Native →** Select the desired service from the table and click on Load Balancer tab for Configuration. The URL suffix you specify under Health Check will be used by DuploCloud during a rolling upgrade i.e., when a service image is changed, then DuploCloud will take down one container replica at a time and bring up a new one. Once it's running, then the

DuploCloud agent running on the host will make a call to the URL and expects a 200 OK. If it does not get a 200 OK, then the upgrade is paused and the user needs to update with an image that fixes the issue.



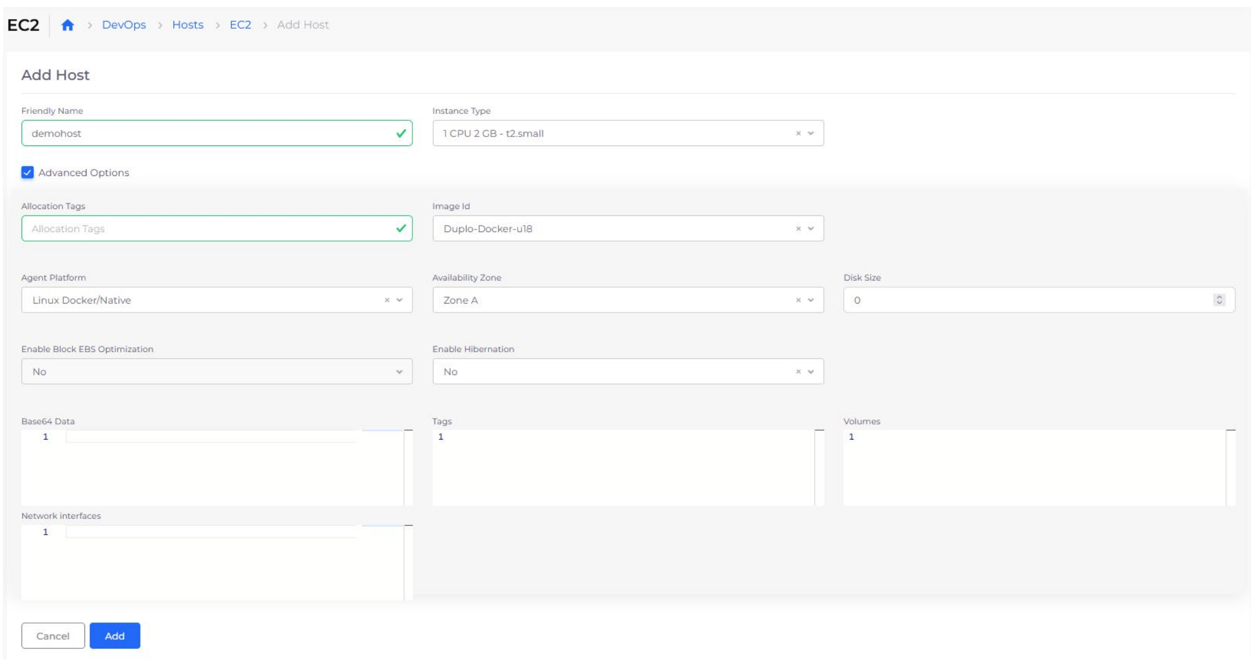
Add Load Balancer Listener

Select Type: Classic
Container port: 8080
External port: 443
Visibility: Public
Application Mode: Docker Mode
Health Check: /
Backend Protocol: HTTP
Certificates: poc.duplocloud.net

- The DNS name for the service will be present in the Services Table. It takes about 5 to 10 minutes for the DNS name to start resolving.
- Update a Service from the menu **DevOps → Containers → EKS/Native →** click on Edit button under Actions column of the Services table or Select the desired service from the table and click on edit option under the Actions menu button.

Virtual Machines

You can create the EC2 instance from DuploCloud. Go to **DevOps → Hosts → EC2 → +Add** button above the table. Choose the desired instance type. The available instance types are set by your administrator or the plan you choose in DuploCloud. If you are not using this host for hosting containers, then set the pool as none. If you want a public IP for the host, then select the public subnet in the list of availability zones.



Add Host

Friendly Name: demohost

Instance Type: 1 CPU 2 GB - t2.small

[✓] Advanced Options

Allocation Tags: Allocation Tags

Image Id: Duplo-Docker-u18

Agent Platform: Linux Docker/Native

Availability Zone: Zone A

Disk Size: 0

Enable Block EBS Optimization: No

Enable Hibernation: No

Base64 Data

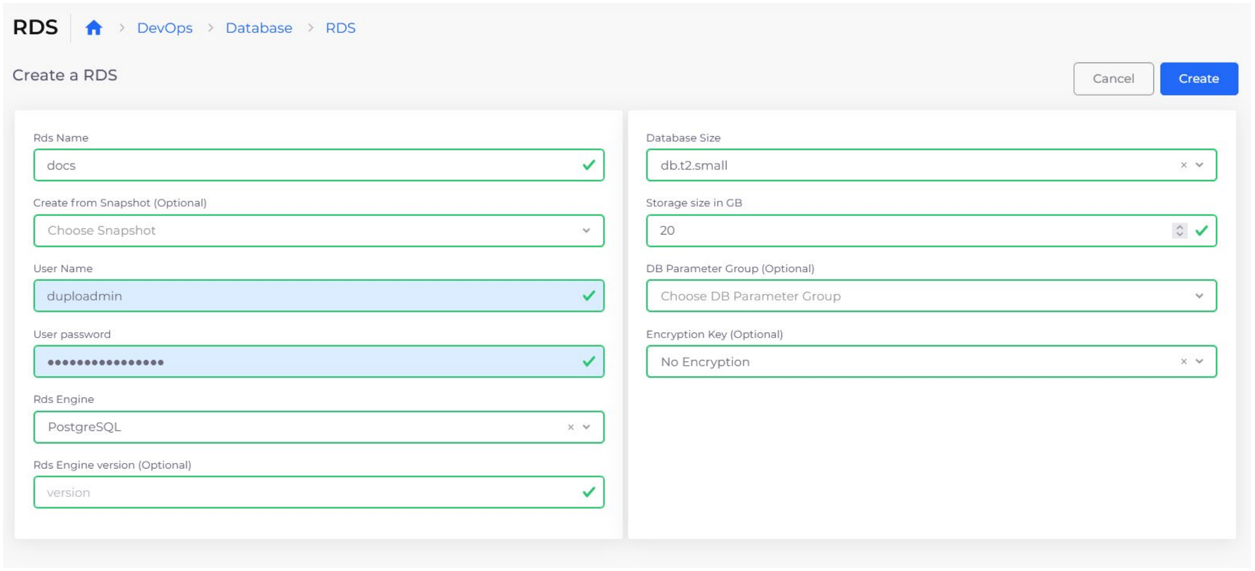
Tags

Volumes

Network interfaces

RDS Database

Create a MySQL, MSFTSQL, PostGRES or Aurora DB from the menu **DevOps** → **Database** → **Relational Databases** → **+Add** button above the table.



Create a RDS

Rds Name: docs

Create from Snapshot (Optional): Choose Snapshot

User Name: duploadmin

User password: ••••••••••••••••

Rds Engine: PostgreSQL

Rds Engine version (Optional): version

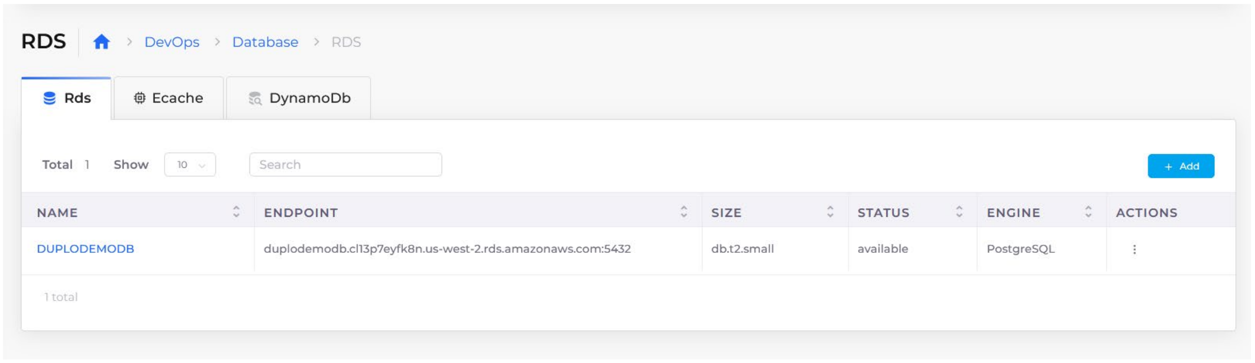
Database Size: db.t2.small

Storage size in GB: 20

DB Parameter Group (Optional): Choose DB Parameter Group

Encryption Key (Optional): No Encryption

Once the DB is created which takes about 5 to 10 mins then you can expand the created DB to fetch the endpoint and credentials. Use this endpoint and credentials to connect to the DB from your application running in the EC2 instance. Note that the DB is accessible only from inside the EC2 instance (including the containers running in it) in the current tenant. *The best way to pass DB endpoint, DBName and credentials to your application is through ENV variables of the Service.*


RDS › DevOps › Database › RDS

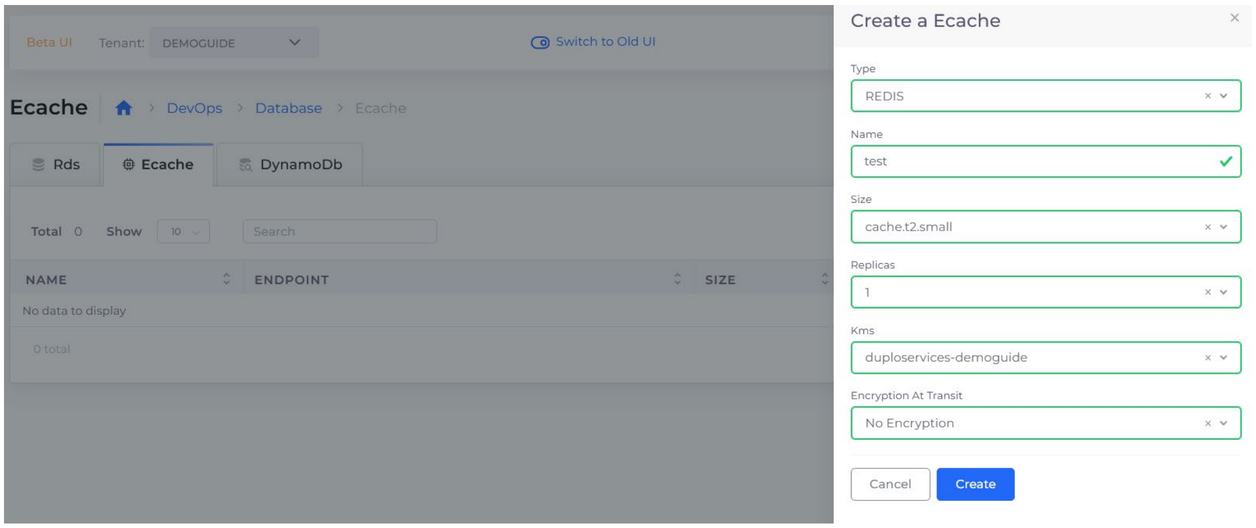
Total 1 Show 10 Search

NAME	ENDPOINT	SIZE	STATUS	ENGINE	ACTIONS
DUPLODEMODB	duplodemodb.cl13p7eyfk8n.us-west-2.rds.amazonaws.com:5432	db.t2.small	available	PostgreSQL	⋮

1 total

Elastic Cache

Create a REDIS or Memcache from **DevOps → Database → Ecache → +Add** button above the table. *The best way to pass the cache endpoint to your application is through ENV variables of the Service.*


Beta UI Tenant: DEMOGUIDE Switch to Old UI

Ecache › DevOps › Database › Ecache

Total 0 Show 10 Search

NAME	ENDPOINT	SIZE
No data to display		

0 total

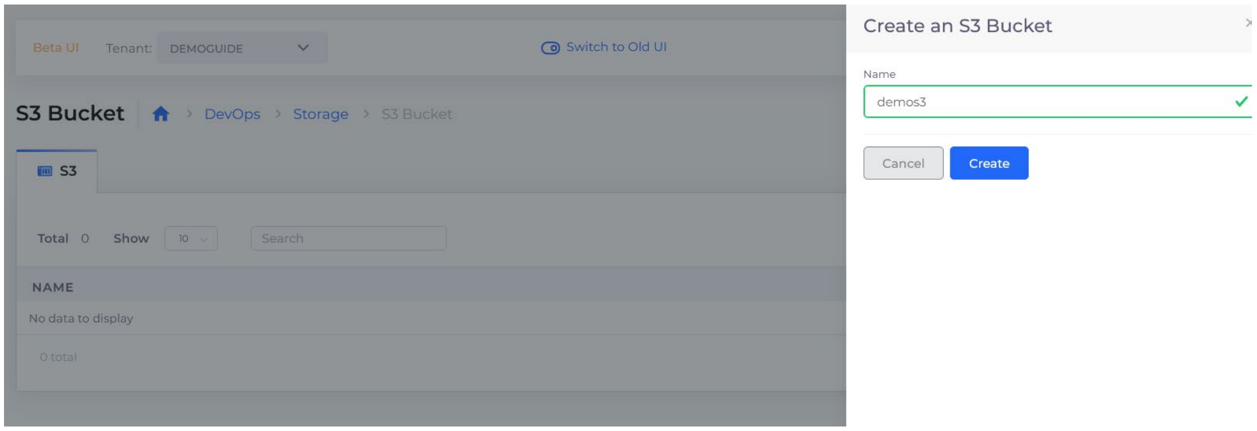
Create a Ecache

Type: REDIS
Name: test
Size: cache.t2.small
Replicas: 1
Kms: duploservices-demoguide
Encryption At Transit: No Encryption

S3 Bucket

Create S3 Bucket from **DevOps → Storage → S3 → +Add** button above the table. Provide the name and select S3Bucket from the dropdown. The required permissions to access the S3 bucket from VM, Lambda functions and containers is provisioned automatically through Instance profiles and hence no Access key is required in the Application code. Application code should use the IAM role/Instance profile approach to connect to these services i.e., the AWS SDK constructor which only uses region should be used. Uploading the file and changing the bucket permissions can be done directly in the AWS console which can be accessed by clicking on the

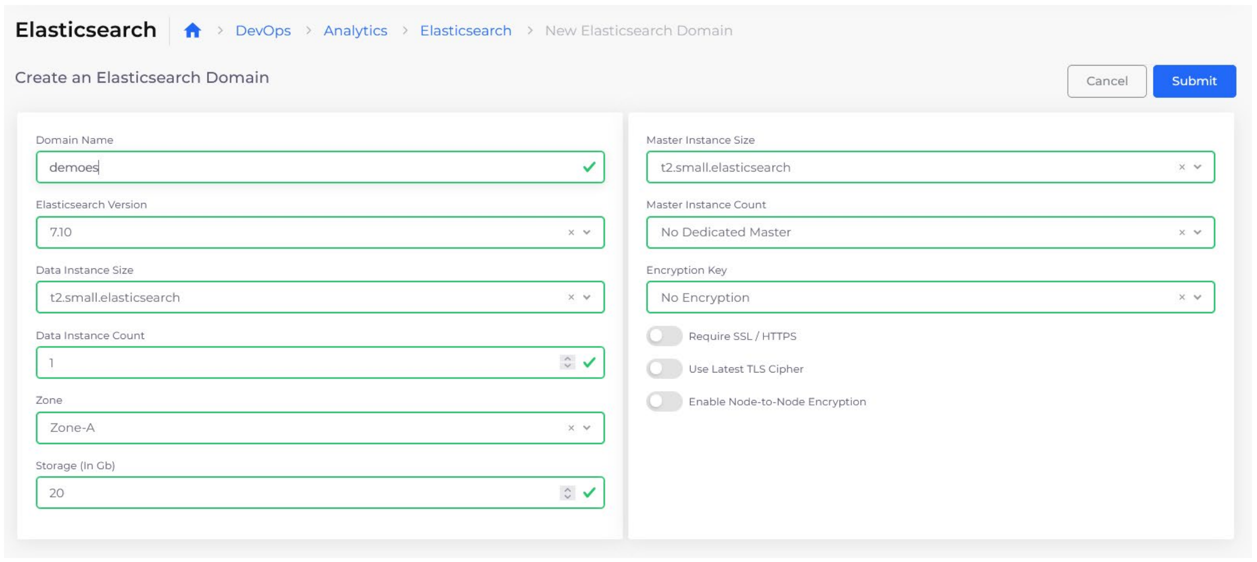
>_ icon. Within this login the user will have enough permissions to configure the bucket, but no access or security level permissions will be given.



Name: demos3

Elastic Search

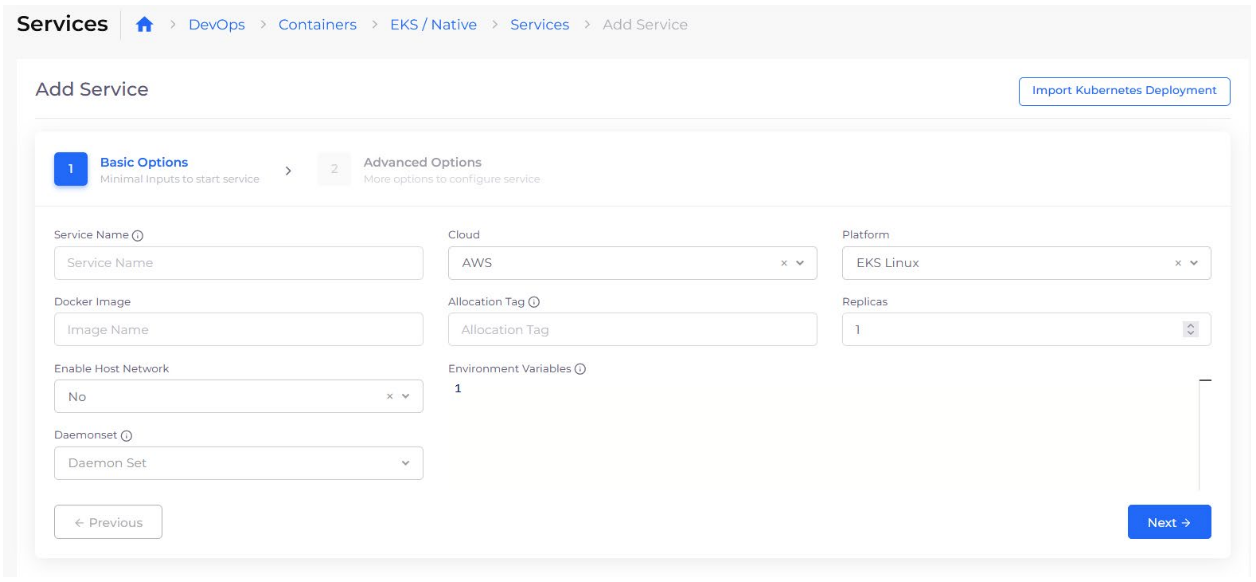
Create an Elasticsearch domain from **DevOps → Analytics → Elasticsearch → +Add** button above the table.


Elasticsearch
DevOps > Analytics > Elasticsearch > New Elasticsearch Domain
Create an Elasticsearch Domain
Cancel
Submit

Domain Name	demoes
Elasticsearch Version	7.10
Data Instance Size	t2.small.elasticsearch
Data Instance Count	1
Zone	Zone-A
Storage (In Gb)	20
Master Instance Size	t2.small.elasticsearch
Master Instance Count	No Dedicated Master
Encryption Key	No Encryption
Require SSL / HTTPS	off
Use Latest TLS Cipher	off
Enable Node-to-Node Encryption	off

Containers

- **Built-In:** Any docker container can be deployed in the VM. Go to **DevOps → Containers → EKS/Native → +Add** button above the table. Give a name for your services (no spaces); number of replicas; Docker image; volumes (if any). The number of replicas must be less than or equal to the number of hosts in the fleet. You can use AllocationTags to deploy the container in a specific set of hosts.


Services > DevOps > Containers > EKS / Native > Services > Add Service

Add Service

Import Kubernetes Deployment

1 Basic Options — Minimal inputs to start service
2 Advanced Options — More options to configure service

Service Name: Service Name
Cloud: AWS
Platform: EKS Linux
Docker Image: Image Name
Allocation Tag: Allocation Tag
Replicas: 1
Enable Host Network: No
Environment Variables: 1
Daemonset: Daemon Set

← Previous   Next →

- **EKS:** DuploCloud supports EKS and AKS out of box. Kubernetes clusters are created during Infrastructure setup under **Administrator → Infrastructure**. The cluster is created in the same VPC as the Infrastructure. Typically, it takes 10 minutes for an Infrastructure with AKS/EKS cluster to be ready. Next, we will walk you through deploying an application within a Tenant in Kubernetes. The application will comprise of set of VMs, Deploymentset(pods) with an application load balancer. Pods can be deployed either through the DuploCloud Portal or through Kubectl(HelmCharts)

    - **Adding Tenants:** We map each tenant to a namespace in Kubernetes. For example, if a tenant is called analytics in DuploCloud, then there will be a namespace called duploservices-analytics in Kubernetes. All components of an application within this tenant are placed in this namespace. Since Nodes cannot be part of namespace in Kubernetes, we set a label called "tenantname" for all the nodes that are launched within the tenant. For example, a node launched in the analytics tenant will have label called "tenantname: duploservices-analytics". Any pods that are launched through the DuploCloud UI will have appropriate nodeselector to tie the pod to the nodes within the tenant. If you are deploying directly via kubectl then make sure your deployment has the appropriate nodeselector.

    - **Adding Hosts:** Once the tenant is created. You can go to **DevOps** and change to the particular tenant. Under the hosts menu, start by creating new Nodes(Hosts). Make sure the value of pool is set to "Eks Linux" (which will be the default).

- **Docker registry credentials & Kubernetes secrets:** These can be set **DevOps → Containers → EKS/Native** tabs. Docker registry credentials are passed to the Kubernetes cluster as kubernetes.io/dockerconfigjson. Other Kubernetes secrets can also be set here and referenced in your deployment later.

- **Adding Services:** When we say services, we don't mean a Kubernetes service, but we mean the DuploCloud term service (See terminology at the top). You can deploy DuploCloud services by clicking **+Add** button under the **EKS/Native** tab. Implicitly, DuploCloud converts these services either into a deployment set or a stateful set. If there are no volume mappings, then it is a deployment set, otherwise it is a stateful set. Most configs are self-explanatory, for example, Images, Replicas and environmental variables. The other advanced configuration can be provided under the other K8 Config section. The content of this section is a 1-1 mapping of the Kubernetes API. All types of configurations for deployments are stateful sets and are supported, while putting the appropriate JSON under the other K8 Config section. For example, to reference the secrets via config map, the JSON would look like the following.

```
{
    "Volumes": [
            {
                    "name": "config-volume",
                    "configMap": {
                            "name": "game-config"
                    }
            }
    ],
    "VolumesMounts": [
            {
                    "name": "config-volume",
                    "mountPath": "/etc/config"
            }
    ]
}
```

- **Kubectl token & config:** DuploCloud provides you JIT token(15min) to access the kubectl cluster. You can get the temporary token and the k8 cluster URL from the Infrastructure you created. Select the **Infrastructure → EKS** tab. Once you have the token and URL run these commands in your local.

```
kubectl config --kubeconfig=config-demo set-cluster EKS_CLUSTER --server=[EKS_API_URL] --insecure-skip-tls-verify

kubectl config --kubeconfig=config-demo set-credentials tempadmin --token=[TOKEN]
kubectl config --kubeconfig=config-demo set-context EKS --
cluster=EKS_CLUSTER --user=tempadmin --namespace=duploservices-
[TENANTNAME]
```

```
export KUBECONFIG=config-demo
```

```
kubectl config use-context EKS
```

With these commands you have configured kubectl to point and access the k8 cluster. You can now run the commands to apply the deployment templates. `kubectl apply -f nginx.yaml`

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment-g
  labels:
    app: nginx-deployment-g
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx-deployment-g
  template:
    metadata:
      labels:
        app: nginx-deployment-g
    spec:
      nodeSelector:
        tenantname: "duploservices-stgeast1"
      containers:
      - name: nginx
        image: nginx:latest
        ports:
        - containerPort: 80
```

If you want longer duration tokens you can create them on your own and secure them out of band from DuploCloud.

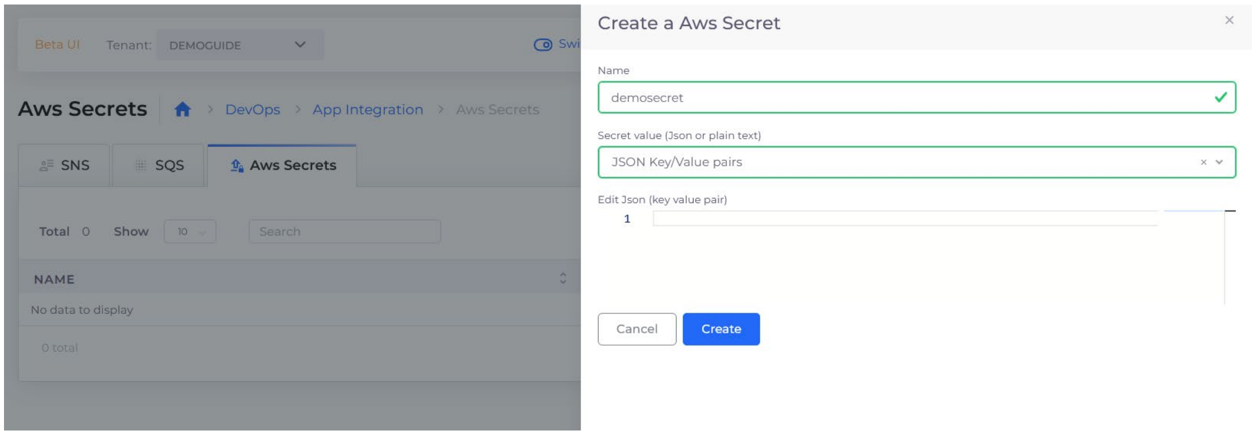
- **Services display:** Once the deployment command rruns successfully, go to the **services** tab of the Tenant. You can see those deployments. Now you can attach the load balancers for the services.

- ○ **Load balancers, WAF and other workflows:** These all remain the same. See above on creation of these.

- **ECS Fargate:** Documentation TBD. Please contact the DuploCloud team for assistance.

Secrets Manager

Create any secrets under **DevOps → App Integration → Aws Secrets → +Add** button above the table.



Name: demosecret

Secret value (Json or plain text): JSON Key/Value pairs

Edit Json (key value pair)

DynamoDB

Create DynamoDB from under **DevOps → Database → DynamoDb tab → +Add** button above the table. The required permissions to access the DynamoDB from VM, Lambda functions and containers are provisioned automatically through Instance profiles and hence no Access key is required in the Application code. Application code should use the IAM role/Instance profile approach to connect to these services i.e., the AWS SDK constructor which only uses region should be used. More detailed configuration of the DynamoDB can be done directly in the AWS console which can be accessed by clicking on the Console >_ icon. Within this login the user will have enough permissions to configure the desired application specific details of the DynamoDB. But no access or security level permissions will be given.

SQS Queue

Create SQS Queue from under **DevOps → App Integration → SQS tab → +Add** button above the table. Refer to DynamoDB to know more about the permissions.

SNS Topic

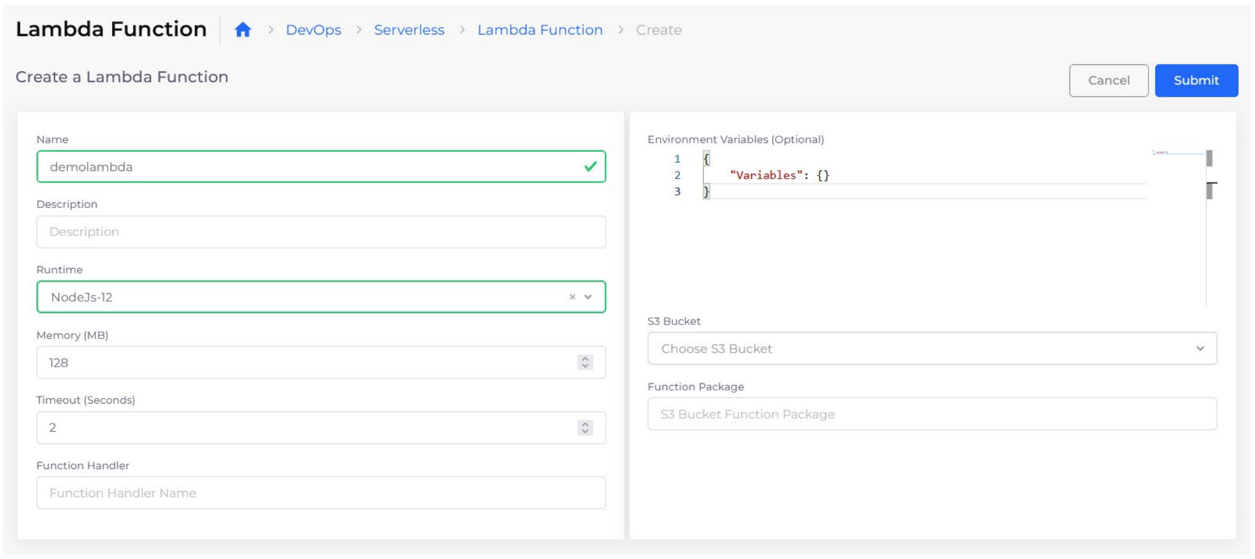
Create SNS Topic from under **DevOps → App Integration → SNS tab → +Add** button above the table. Refer to DynamoDB to know more about the permissions.

Kinesis Stream

Create Kinesis Stream from under **DevOps → Analytics → Kinesis tab → +Add** button above the table. Refer to DynamoDB to know more about the permissions.

Lambda

Create Lambda function from under **DevOps → Serverless → Lambda tab → +Add** button. Give a name for the Lambda function and other values. This will create the lambda function. Click on AWS console to go to the AWS console for this function. Test the function. You can look at the tutorial above for the same or look at the AWS documentation.



Name: demolambda
Description: Description
Runtime: NodeJs-12
Memory (MB): 128
Timeout (Seconds): 2
Function Handler: Function Handler Name
Environment Variables (Optional)

```
{
    "Variables": {}
}
```

S3 Bucket: Choose S3 Bucket
Function Package: S3 Bucket Function Package

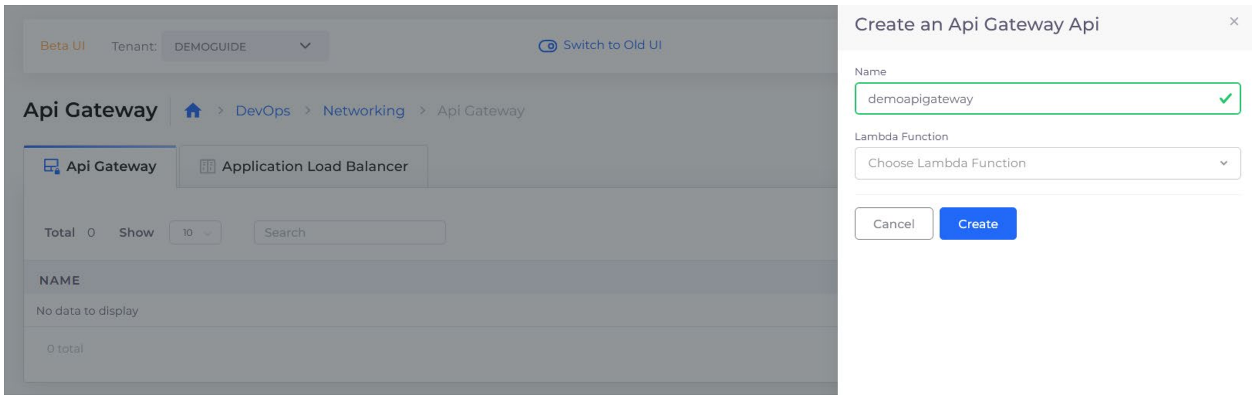
- **Updating Lambda Function and configuration:** To update the code for the Lambda function, create a new package with a different name and upload in S3 bucket again. Then select the lambda function in the table and click Edit under dropdown menu in Actions column. Make sure the right S3 bucket has been selected and provide the name of the function. To update a function configuration like timeout memory, etc. use the edit configuration button.

- **Integrating with other resources:** DuploCloud enables you to create a classic micro-services based architecture where your Lambda function can integrate with any other resource within your tenant like S3, Dynamo, RDS or other Docker based microservices. DuploCloud will implicitly enable the Lambda function to communicate with other resources but block any communication outside the tenant (except ELB).

- **Triggers and Event Sources:** To setup a trigger or event source, the resource needs to be created via the DuploCloud Portal. Subsequently, one could trigger directly from that resource to the Lambda function in the AWS console menu of your Lambda function. Resources could be S3 buckets, API Gateway, DynamoDB, SNS etc. An example trigger via API Gateway was described in the tutorial.
- **Passing Secrets:** Passing secrets to a Lambda function can be done in much the same way as passing secrets to your Docker based service i.e., using environmental variables. For example, you can create a SQL database from the **Database → RDS** menu in DuploCloud, provide username and password, then in the Lambda menu give the same username and password. No secrets need to be stored anywhere outside like vault, git repo, etc.

API Gateway

AWS ApiGateway RestApi is created from the DuploCloud Portal which will take care of creating the security policies to make the API Gateway accessible to other resources (like Lambda function) within the tenant. From the DuploCloud Portal only create the RestApi. All other configuration for the API (like defining methods, resources and pointing to lambda functions) should be done in the AWS console. The console for the API can be reached by selecting the API in **DevOps → Networking → API Gateway** table and then clicking on the Console button under the Actions menu. The above tutorial displays the integration of API Gateway and lambda function.



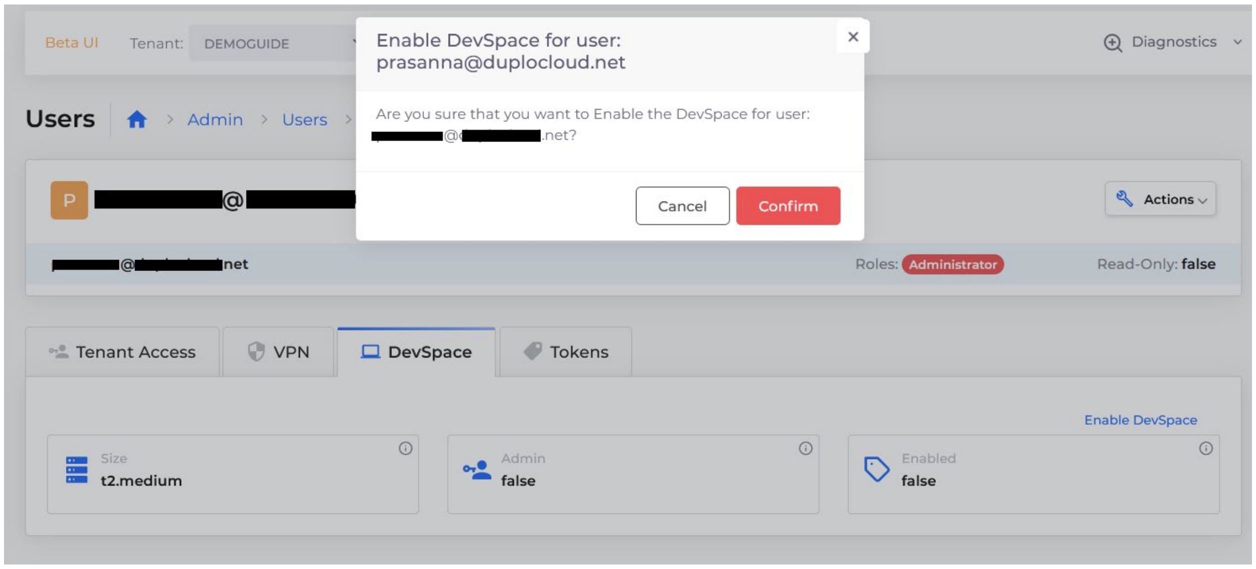
Create an Api Gateway Api

Name: demoapigateway

Lambda Function: Choose Lambda Function

Workspaces

Individual workspaces/Devspaces can be created for the user in DuploCloud. These workspaces will have access to all the resources in the tenant which user has access. To create a workspace go to **Administrator → Users → Select user → Devspace** tab.


Enable DevSpace for user: prasanna@duplocloud.net

Are you sure that you want to Enable the DevSpace for user: [redacted].net?

Cancel | Confirm

Users > Admin > Users

[redacted]

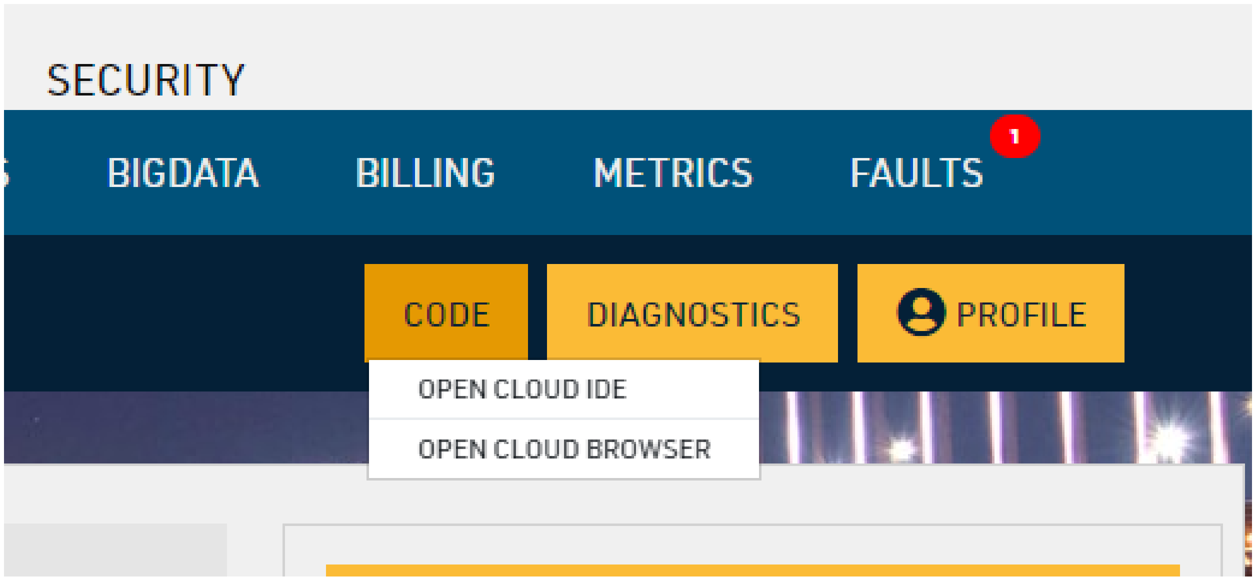
[redacted]net — Roles: Administrator — Read-Only: false

Tenant Access | VPN | DevSpace | Tokens

Enable DevSpace

Size	Admin	Enabled
t2.medium	false	false

Once the workspace is created, it can be accessed from the Code next to Diagnostics.



Service Discovery (CloudMap)

Documentation TBD. Please contact DuploCloud team for assistance.

IoT

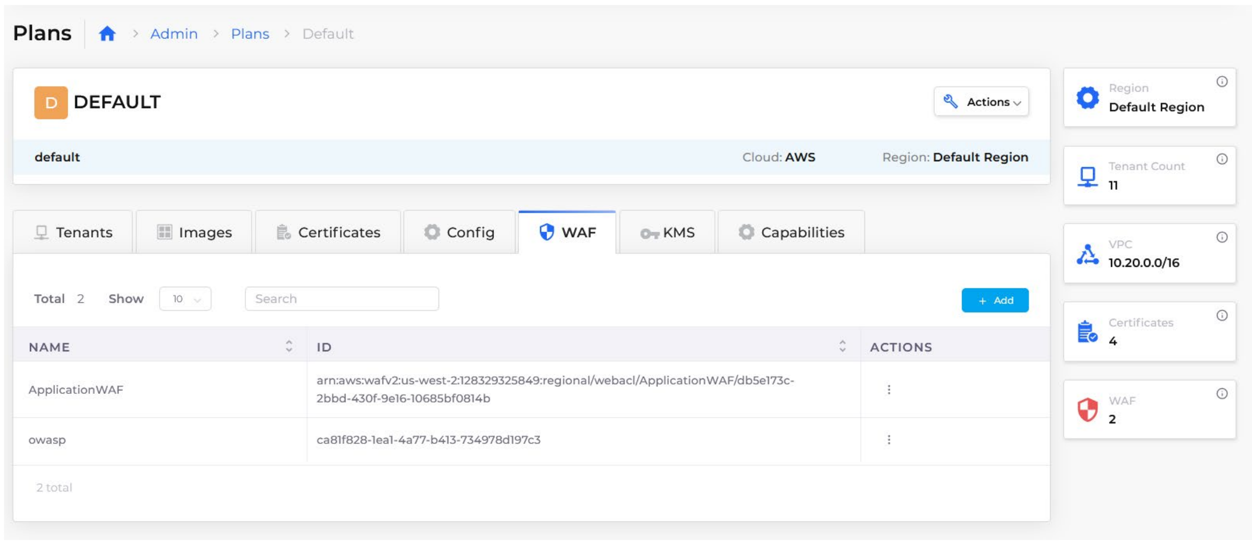
Documentation TBD. Please contact DuploCloud team for assistance.

Cloud Watch

By default, the metrics for a resource are available in the Metrics page in DuploCloud. But some metrics need agents to be installed in the system to collect the information, such as AWS Agent. DuploCloud provides a way to automatically install these agents on all the hosts whenever they are provisioned. For more information, refer to https://portal.duplocloud.net/compliance/Implementation.html#agent-modules

WAF

Creation of a Web Application Firewall is a onetime process. Create a WAF in the AWS Console, fetch the ID/ARN and update the plan in DuploCloud. Once the plan has the WAF it can be used attached to the LB.


Plans > Admin > Plans > Default

DEFAULT

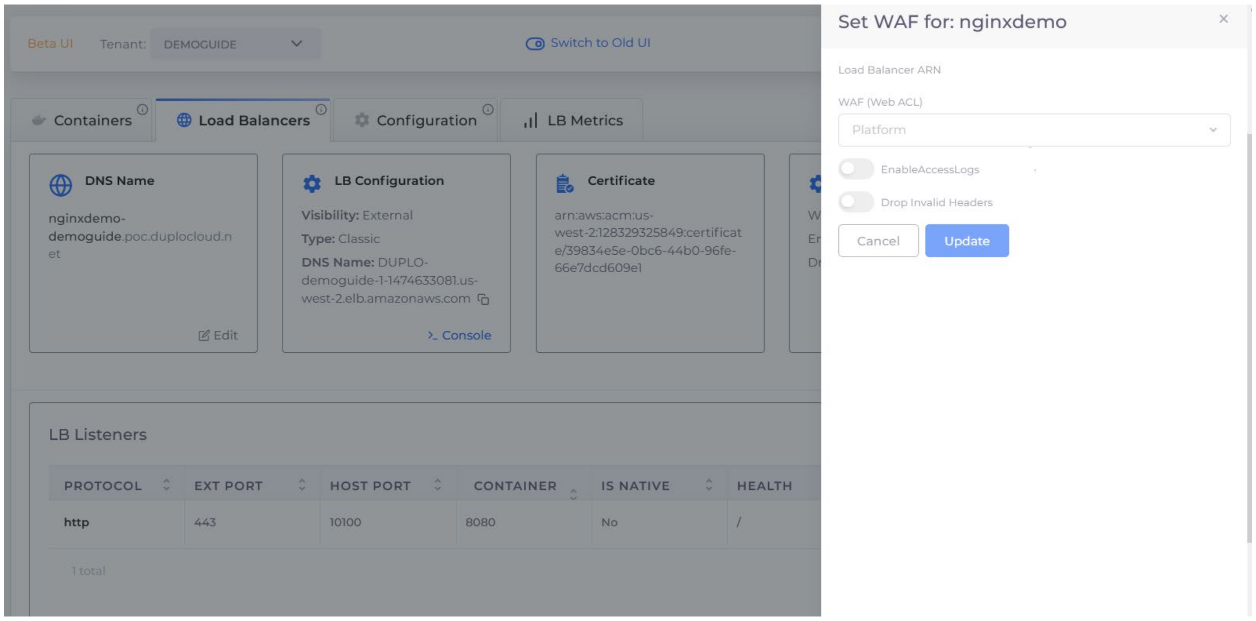
default Cloud: AWS Region: Default Region

Region: Default Region; Tenant Count: 11; VPC: 10.20.0.0/16; Certificates: 4; WAF: 2

Total 2

NAME	ID	ACTIONS
ApplicationWAF	arn:aws:wafv2:us-west-2:128329325849:regional/webacl/ApplicationWAF/db5e173c-2bbd-430f-9e16-10685bf0814b	⋮
owasp	ca81f828-1ea1-4a77-b413-734978d197c3	⋮

2 total


Set WAF for: nginxdemo

Load Balancer ARN

WAF (Web ACL): Platform

EnableAccessLogs

Drop Invalid Headers

Tenant: DEMOGUIDE

DNS Name: nginxdemo-demoguide.poc.duplocloud.net

LB Configuration — Visibility: External; Type: Classic; DNS Name: DUPLO-demoguide-1-1474633081.us-west-2.elb.amazonaws.com

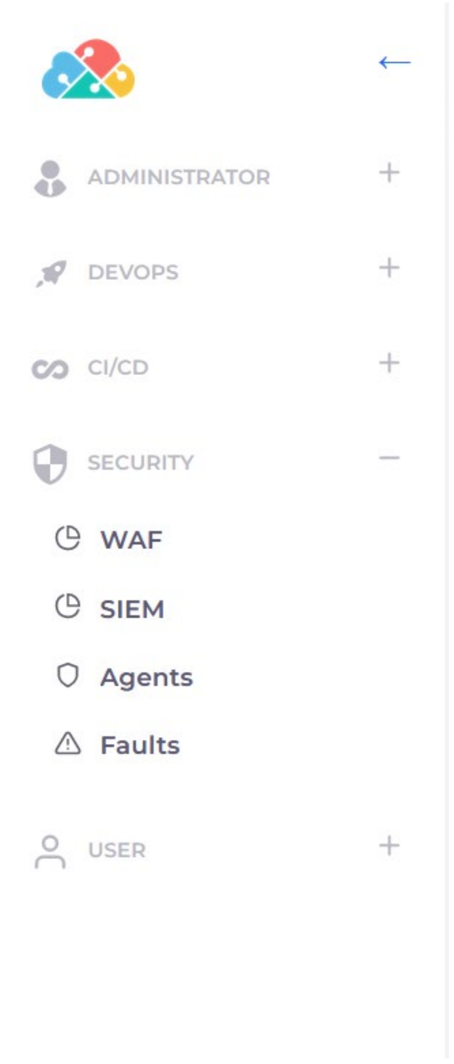
Certificate: arn:aws:acm:us-west-2:128329325849:certificate/39834e5e-0bc6-44b0-96fe-66e7dcd609e1

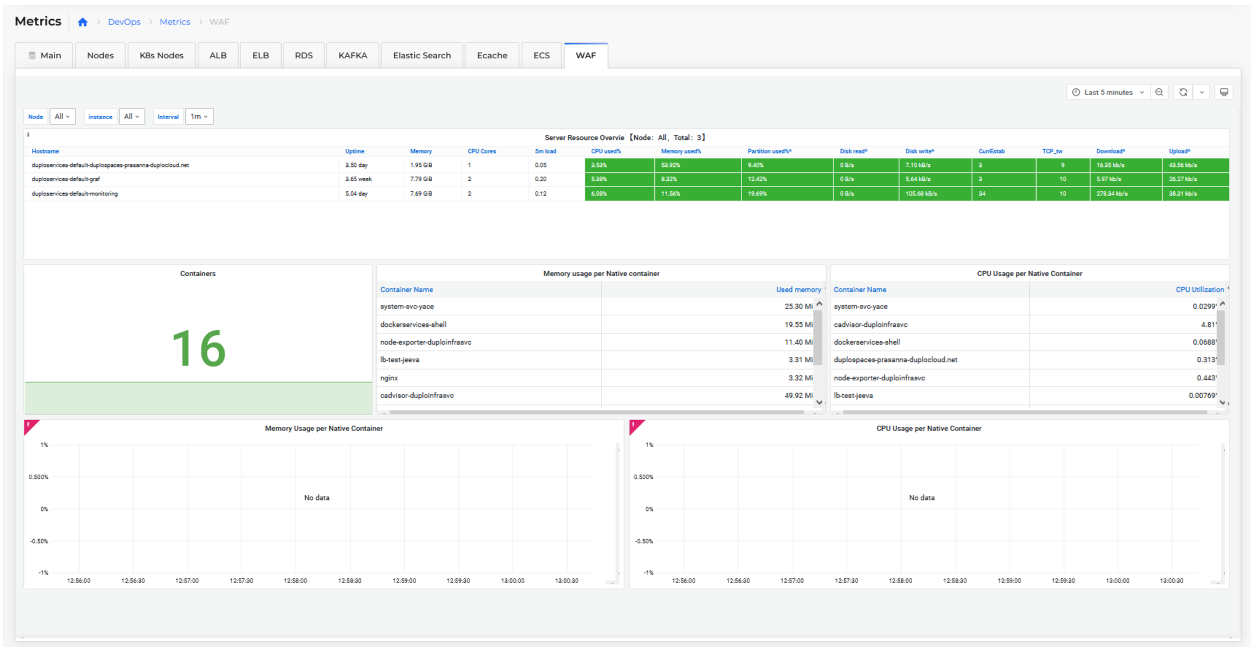
LB Listeners

PROTOCOL	EXT PORT	HOST PORT	CONTAINER	IS NATIVE	HEALTH
http	443	10100	8080	No	/

1 total

DuploCloud also provides a WAF Dashboard through which you can analyze the traffic that is coming in and the requests that are blocked. The Dashboard can be accessed from **Security → WAF**.


ADMINISTRATOR
DEVOPS
CI/CD
SECURITY
WAF
SIEM
Agents
Faults
USER



Metrics › DevOps › Metrics › WAF

Main | Nodes | K8s Nodes | ALB | ELB | RDS | KAFKA | Elastic Search | Ecache | ECS | WAF

Last 5 minutes

Node: All | instance: All | Interval: 1m

Server Resource Overvie【Node: All, Total: 3】

Hostname	Uptime	Memory	CPU Cores	5m load	CPU used%	Memory used%	Partition used%*	Disk read*	Disk write*	CurrEstab	TCP_tw	Download*	Upload*
duploservices-default-duplospaces-prasanna-duplocloud.net	3.50 day	1.95 GiB	1	0.05	3.53%	53.92%	9.40%	0 B/s	7.10 kB/s	3	9	16.35 kb/s	43.56 kb/s
duploservices-default-graf	3.65 week	7.79 GiB	2	0.20	5.38%	8.32%	12.42%	0 B/s	5.64 kB/s	3	10	5.97 kb/s	26.27 kb/s
duploservices-default-monitoring	5.04 day	7.69 GiB	2	0.12	6.08%	11.56%	19.69%	0 B/s	105.68 kB/s	34	10	278.34 kb/s	38.31 kb/s

Containers

16

Memory usage per Native container

Container Name	Used memory
system-svc-yace	25.30 Mi
dockerservices-shell	19.55 Mi
node-exporter-duploinfrasvc	11.40 Mi
lb-test-jeeva	3.31 Mi
nginx	3.32 Mi
cadvisor-duploinfrasvc	49.92 Mi

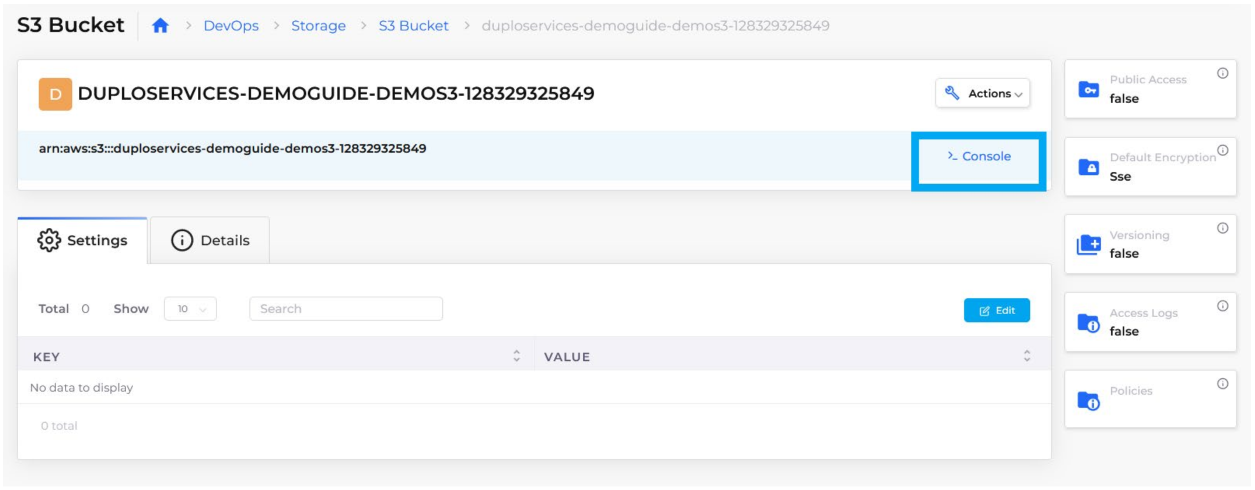
CPU Usage per Native Container

Container Name	CPU Utilization
system-svc-yace	0.0299
cadvisor-duploinfrasvc	4.81
dockerservices-shell	0.0688
duplospaces-prasanna-duplocloud.net	0.313
node-exporter-duploinfrasvc	0.443
lb-test-jeeva	0.00769

Memory Usage per Native Container
No data
CPU Usage per Native Container
No data

Using AWS Console

DuploCloud users get AWS console access for advanced configurations of S3, Dynamo, SQS, SNS Topic, Kinesis stream, and API Gateway resources that were created through DuploCloud. ELB and EC2 parts of console is not supported yet. You can click on the AWS console icon to take you directly to that resource in AWS console with permissions scoped to the current tenant/project. You don't have permissions to create new resources directly from the AWS console. You need to do them in DuploCloud. You can do all operations on resources already created via DuploCloud. For example, you can create an S3 bucket from the DuploCloud UI and go to AWS console to add remove files, setup static web hosting etc. Similarly, you can create a DynamoDB in DuploCloud and use AWS console to add/remove entries in the table.



S3 Bucket › DevOps › Storage › S3 Bucket › duploservices-demoguide-demos3-128329325849

DUPLOSERVICES-DEMOGUIDE-DEMOS3-128329325849

arn:aws:s3:::duploservices-demoguide-demos3-128329325849

Public Access	false
Default Encryption	Sse
Versioning	false
Access Logs	false
Policies	

Total 0 Show 10

KEY	VALUE

No data to display

0 total

FAQ

- **How do I ssh into the host?**
  Under each host you can click on Connection details under Action dropdown which will provide the key file and instructions to ssh.

- **My host is windows how do i RDP?**
  Under host click on Connection details under Action dropdown, it will provide the password and instructions to rdp.

- **How do I get into the container where my code is running?**
  Under the services Status TAB find the host where the container is running. Then ssh into the host (see instructions above) and then run "`sudo docker ps`" and get ID the container. Then run "`sudo docker exec -it containerid bash`". You can tell which one is your container by using the image id. Don't forget the sudo in docker commands

- **I cannot connect to my service URL, how do I debug?**
  Make sure the DNS name is resolving by running in your computer "`ping `". Then we need to check if the application is running by testing the same from within the container. Then ssh into the host and then connect to your docker container using docker exec command (see above). From inside the container CURL the application URL using the IP 127.0.0.1 and port where the application is running. Confirm that this works. Then CURL the same URL using the IP address of the container instead of 127.0.0.1. The IP address can be obtained by running ifconfig command in the container.

  If the connection from within the container is fine, then exit from the container into the host. Now CURL the same endpoint from the host i.e., using container IP and port. If this works then under the ELB UI in DuploCloud note down the host port that DuploCloud created for the given container endpoint. This will be in the range 10xxx or the same as container port. Now try connecting to the "HostIP" and `DuploMappedHostPort` just obtained. If this works as well but the service URL is still failing, contact your enterprise admin or duplolive-support@duplocloud.net.

- **What keys should I use in my application to connect to the AWS resources I have created in DuploCloud like S3, Dynamo, SQS, etc.?**
  You don't need any. Use the AWS constructor that takes only the region as the argument (us-west-2). DuploCloud setup links your instance profile and the resources. The host in DuploCloud already has access to the resources within the same tenant\project.

DuploCloud AWS resources are reachable only from DuploCloud Hosts on the same account. YOU CANNOT CONNECT TO ANY DUPLOCLOUD AWS RESOURCE FROM YOUR LAPTOP.

- **What is rolling upgrade and how do I enable it?**
  If you have multiple replicas in your service i.e., multiple containers then when you update your service like change an image or env variable then DuploCloud will make this change one container at a time i.e., it will bring down the first container and bring up the new one on that host with the updated config. If the new container fails to start or the health check URL set by the user is not returning 200 OK, then DuploCloud will pause the upgrade of remaining containers. A user must intervene by fixing the issue typically by updating the service with a newer image that has a fix. If no health check URL is specified, then DuploCloud only checks for the new container to be running. To specify health check, go the ELB menu and you will find the health check URL suffix.

- **I want to have multiple replicas of my mvc service, how do I make sure that only one of them runs migration?**
  Enable health check for your service and make sure that the API does not return 200 OK till migration is done. Since DuploCloud waits for health check to be complete before upgrading the next service it is guaranteed that only one instance runs migration.

- **One or more of my containers are showing in a pending state, how can I debug?**
  If it is Pending when the desired state is Running, then the image is being downloaded so wait a few minutes. If it's been more than five minutes check the faults from the button below the table. Check if your image name is correct and does not have spaces. Image names are case sensitive so it should be all lower case i.e., the image name in DockerHub should also be lower case.

  If the current state is Pending when desired state is Delete then it means this container is the old version of the service. It is still running as the system is in rolling upgrade and the previous replica is not successfully upgraded yet. Check the faults in other containers of this service for which the Current State is Pending with Desired State as Running…

- **Some of my container status says "pending delete" what does it mean?**
  This means DuploCloud is wanting to remove these containers. The most common reason for this is that another replica of the same service was upgraded but is now not operational. Hence DuploCloud has blocked the upgrade. You might see the other replicas in even "Running" state, but it is possible that health check is failing and that's why the rolling upgrade is blocked. To unblock, fix the service configuration (image, env, etc.) to an error free state.

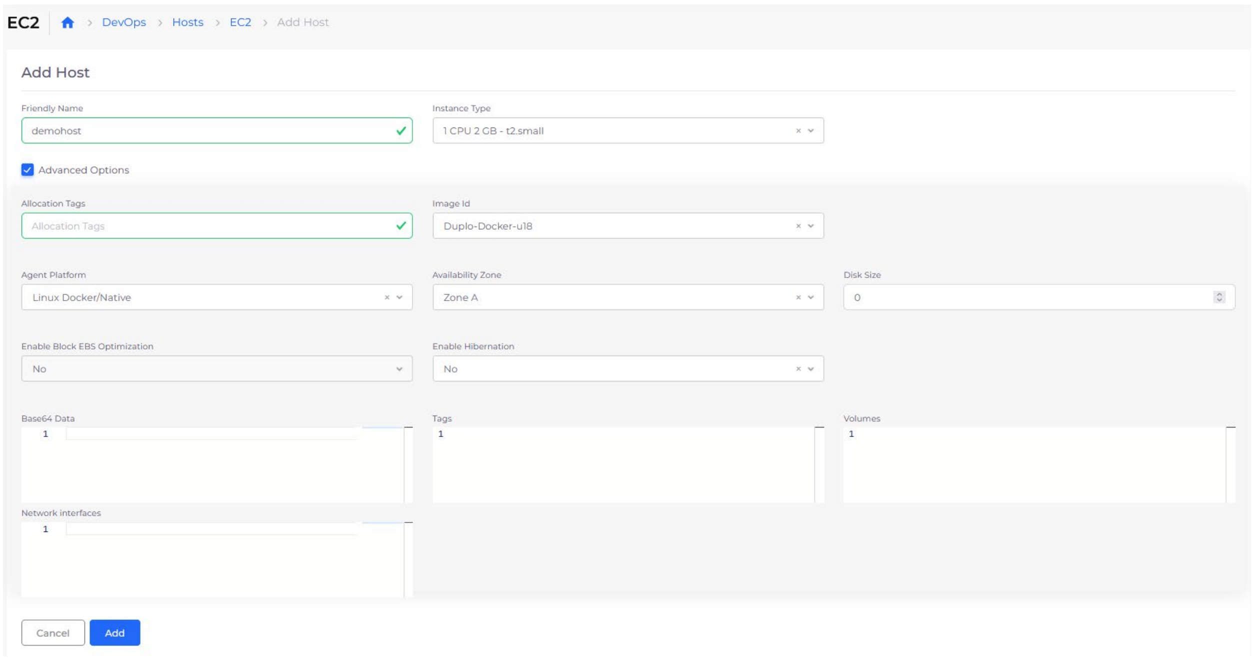
- **How to create host with public IP?**
  While creating host, click on **show advanced** and select the public subnet in the list of availability zone.

AWS Use Cases

Docker Webapp

In this demo, we will deploy a simple Hello World NodeJS web app. DuploCloud pulls Docker images from Docker Hub. You can choose a public image or provide credentials to access your private repository. For the sake of this demo, we will use a ready-made image available on Duplo's repository on Docker Hub.

- Login to your DuploCloud console.
- Select **DevOps → Hosts**, a Host is the instance in which your Docker container will run. You should choose a host with appropriate processing capacity for your application.
- Click on the **+Add** button to choose your host. Fill out the advanced options form if required and click submit.


EC2 > DevOps > Hosts > EC2 > Add Host

Add Host

Friendly Name: demohost
Instance Type: 1 CPU 2 GB - t2.small
[✓] Advanced Options
Allocation Tags: Allocation Tags
Image Id: Duplo-Docker-u18
Agent Platform: Linux Docker/Native
Availability Zone: Zone A
Disk Size: 0
Enable Block EBS Optimization: No
Enable Hibernation: No
Base64 Data
Tags
Volumes
Network interfaces

- You should now see your Host present in the table. Please give it a moment to instantiate.

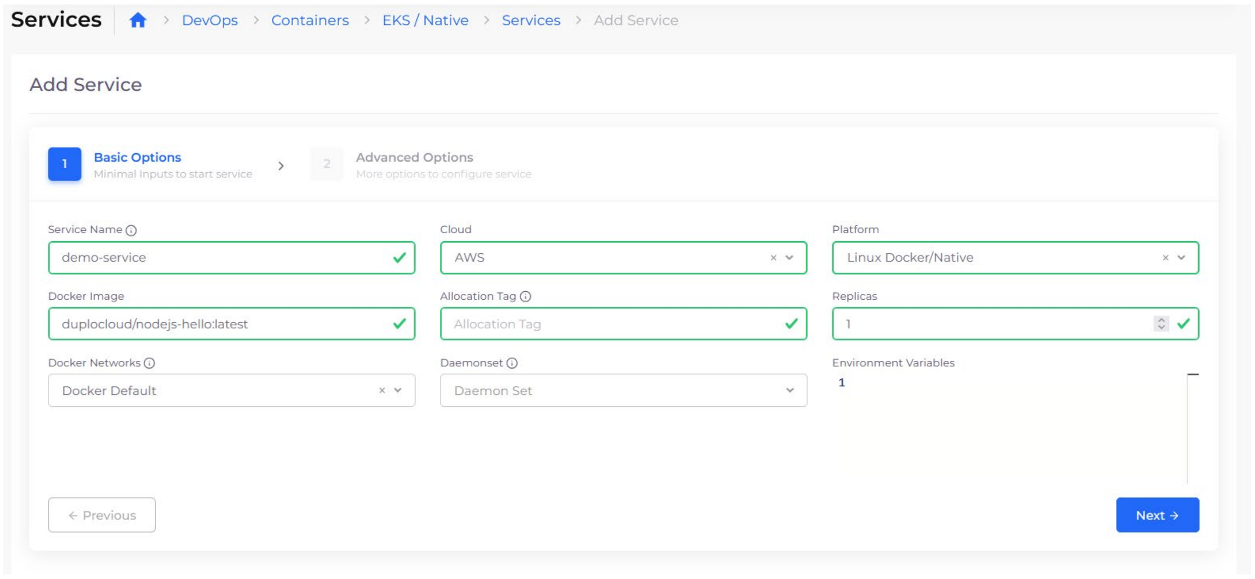
- Next, we can create a Service. A Service is nothing but a container with user specified image and environment variables. Let's go ahead and click **+Add** button to create a new service.

- Name the service "Test-service". For this demo we will use the latest, nodejs-hello image from Duplo's public Docker hub repository. Fill in "`duplocloud/nodejs-hello:latest`" in the Docker Image field.

- Enter the desired number of replicas you want in the swarm. Please note that each replica runs in an individual Host, so the number of replicas must equal the number of Hosts. For the sake of this demo, we will choose 1.

- Fill in the desired environment variables, this is ideal for credentials or application specific configurations.

- Volume mapping is super easy, simply give the host path and container path as shown. Please note that we highly recommend keeping the Hosts stateless and using S3 for static assets. We will keep this field empty for this demo.

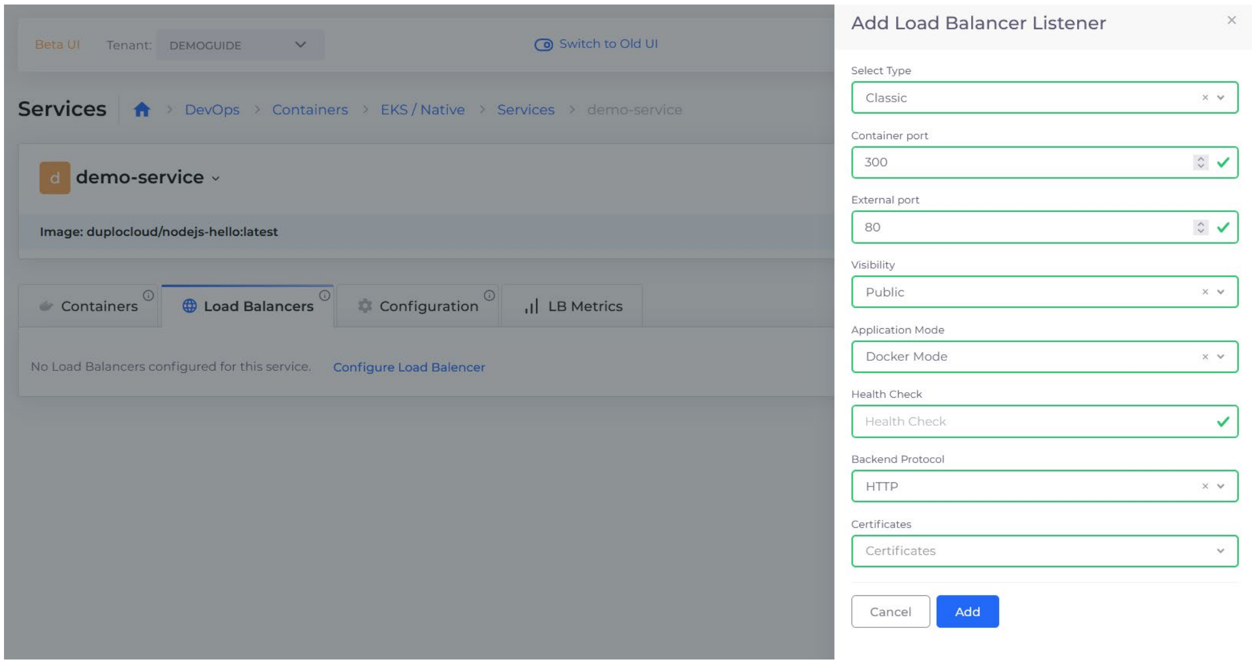

Services
DevOps > Containers > EKS / Native > Services > Add Service
Add Service
1 Basic Options
Minimal inputs to start service
2 Advanced Options
More options to configure service
← Previous
Next →

Service Name: demo-service
Cloud: AWS
Platform: Linux Docker/Native
Docker Image: duplocloud/nodejs-hello:latest
Allocation Tag: Allocation Tag
Replicas: 1
Docker Networks: Docker Default
Daemonset: Daemon Set
Environment Variables: 1

- Hit Create! Please wait a moment for the service to initialize.

- Almost there. Since the hello-nodejs image serves on port 3000 we need to create a load balancer (LB) configuration to map external port (LB) to internal port (container).

- Select the Test-service and click on **configure load balancer** on the load balancer tab. Fill the menu as shown below and click submit. This will also create a DNS name that we can use.


Add Load Balancer Listener

Select Type: Classic

Container port: 300

External port: 80

Visibility: Public

Application Mode: Docker Mode

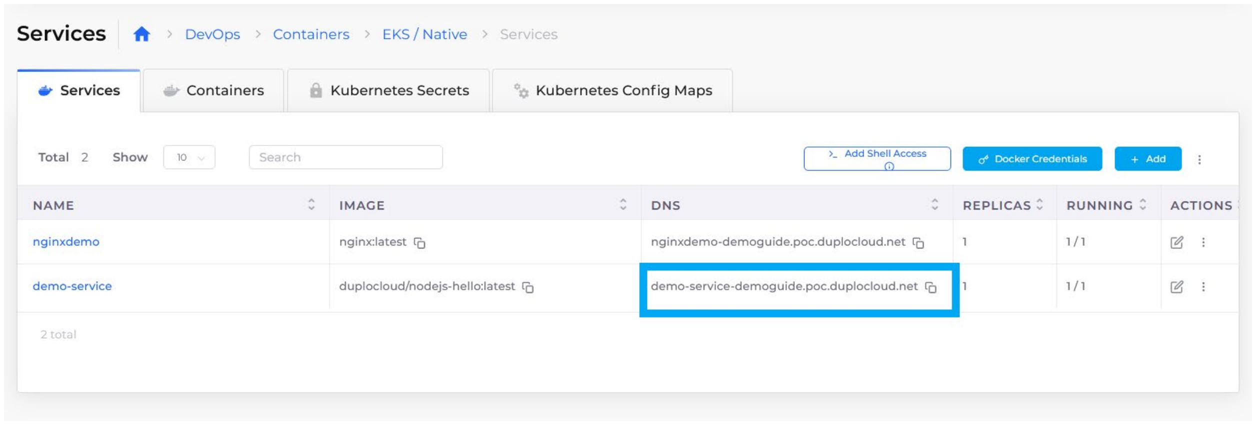
Health Check: Health Check

Backend Protocol: HTTP

Certificates: Certificates

Beta UI
Tenant: DEMOGUIDE
Switch to Old UI
Services
DevOps > Containers > EKS / Native > Services > demo-service
demo-service
Image: duplocloud/nodejs-hello:latest
Containers
Load Balancers
Configuration
LB Metrics
No Load Balancers configured for this service.
Configure Load Balencer
Cancel
Add

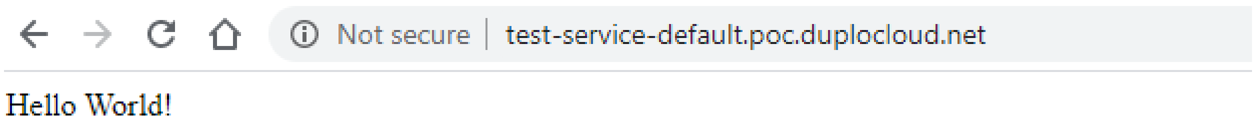
- Please wait for ~5 minutes as it can take a while for the DNS Route table changes to be reflected.



NAME	IMAGE	DNS	REPLICAS	RUNNING	ACTIONS
nginxdemo	nginx:latest	nginxdemo-demoguide.poc.duplocloud.net	1	1 / 1	
demo-service	duplocloud/nodejs-hello:latest	demo-service-demoguide.poc.duplocloud.net	1	1 / 1	

2 total

- Expand Test-service and copy the URL displayed under DNS. Hit the URL in the browser and You should see the Hello world serving our welcome message.


Not secure | test-service-default.poc.duplocloud.net

Hello World!

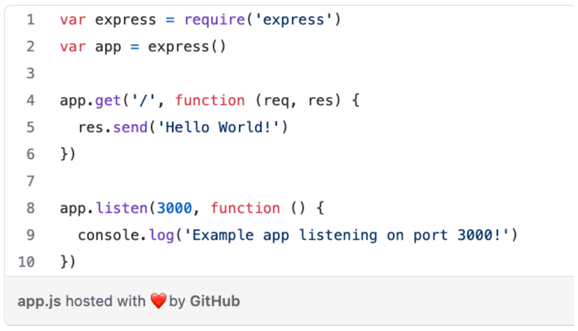
- Congratulations! You have just launched your first web service on Duplo!

S3 Backed Webapp

In the last tutorial we deployed a NodeJS based webserver and accessed it using the DNS name that DuploCloud created for us. In this tutorial we will take it a step further. We will create a text file with a message and upload it to S3. Next, we will modify our NodeJS application to access this file and display the message to every visitor. This purpose of the tutorial is to familiarize yourself with how easy it is to manage resources on AWS using DuploCloud.

- The code in the Docker container that we used in Part 1 can be found on this repo. It's a simple NodeJS web server. Clone this repository and make the following changes.

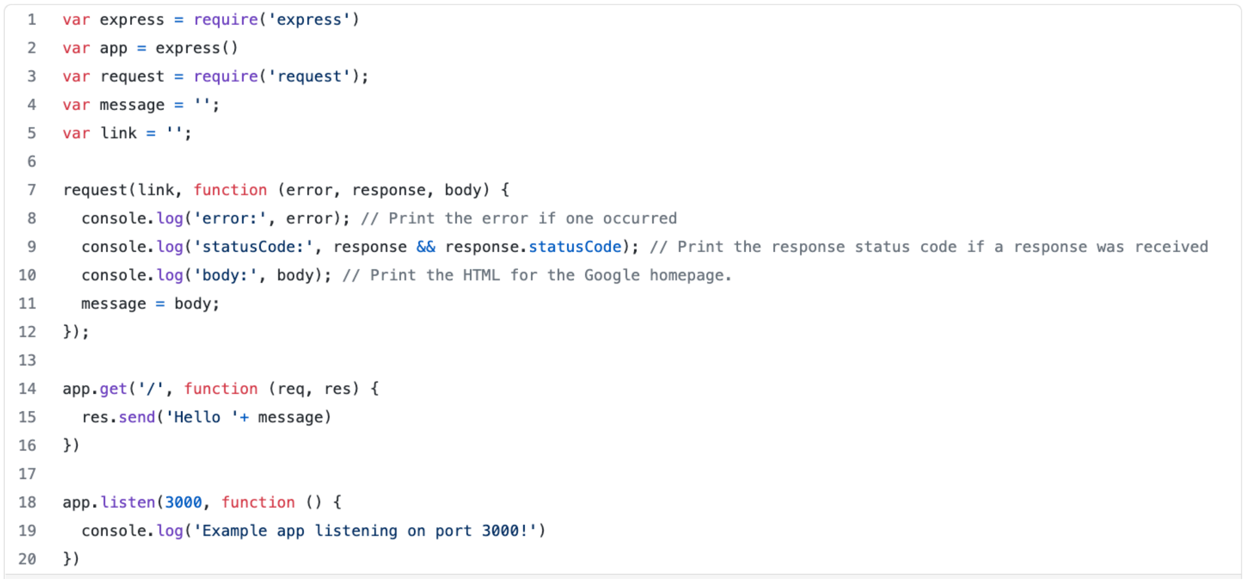
- Current app.js should look like this:



```
var express = require('express')
var app = express()

app.get('/', function (req, res) {
  res.send('Hello World!')
})

app.listen(3000, function () {
  console.log('Example app listening on port 3000!')
})
```

app.js hosted with ❤ by GitHub

- Let’s make the following changes

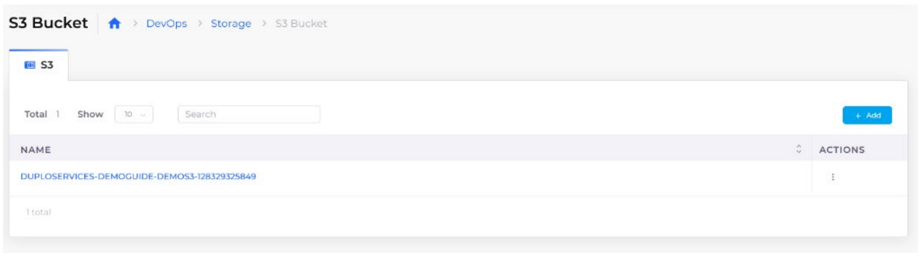


```
var express = require('express')
var app = express()
var request = require('request');
var message = '';
var link = '';

request(link, function (error, response, body) {
  console.log('error:', error); // Print the error if one occurred
  console.log('statusCode:', response && response.statusCode); // Print the response status code if a response was received
  console.log('body:', body); // Print the HTML for the Google homepage.
  message = body;
});

app.get('/', function (req, res) {
  res.send('Hello '+ message)
})

app.listen(3000, function () {
  console.log('Example app listening on port 3000!')
})
```

- In the above code we added a link variable that will point to the S3 bucket that we will create shortly, message variable is the message the text file in the S3 bucket will contain.

- We can now go ahead and create a text file with the message "My Name is Pranay",

- Now that we have this file, we'll upload it to S3. Login to your DuploCloud Console and Navigate to **DevOps → Storage → S3**.

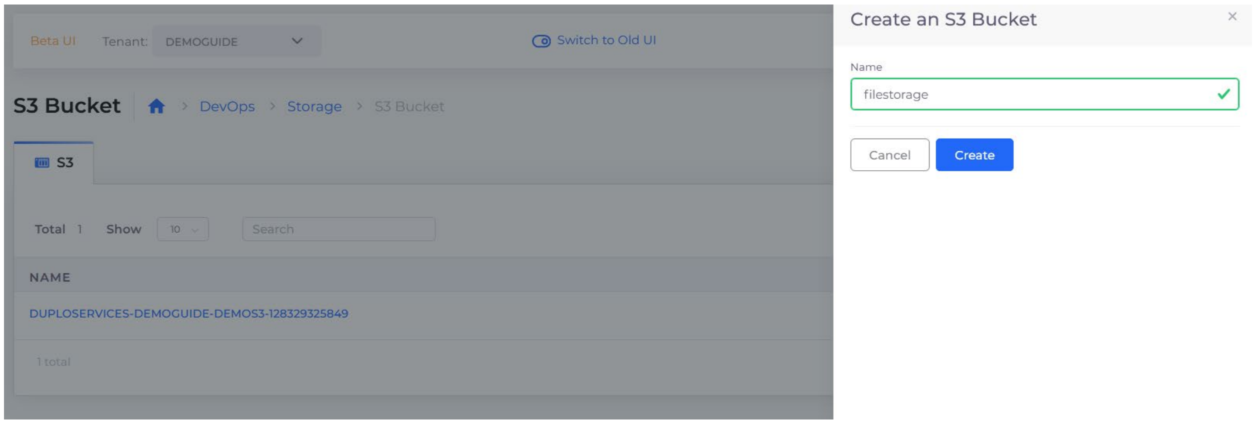

S3 Bucket > DevOps > Storage > S3 Bucket

Total 1 Show 10

NAME	ACTIONS
DUPLOSERVICES-DEMOGUIDE-DEMOS3-128329325849	⋮

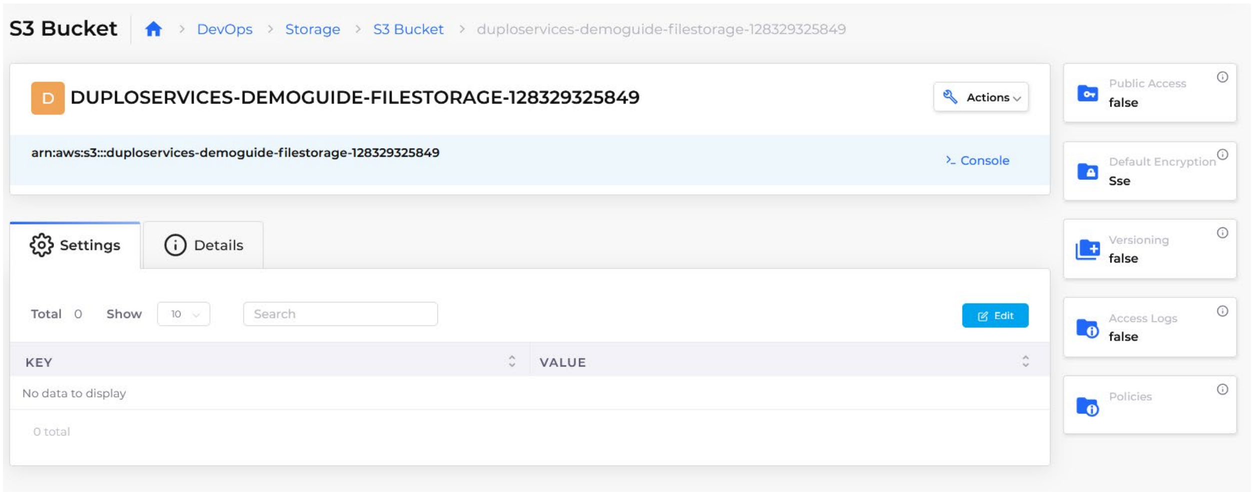
1 total

- Click **+Add** button to create a S3 bucket, provide the resource name and click on Create.


Create an S3 Bucket

Name: filestorage

- This will create a S3 bucket


S3 Bucket > DevOps > Storage > S3 Bucket > duploservices-demoguide-filestorage-128329325849

DUPLOSERVICES-DEMOGUIDE-FILESTORAGE-128329325849

arn:aws:s3:::duploservices-demoguide-filestorage-128329325849

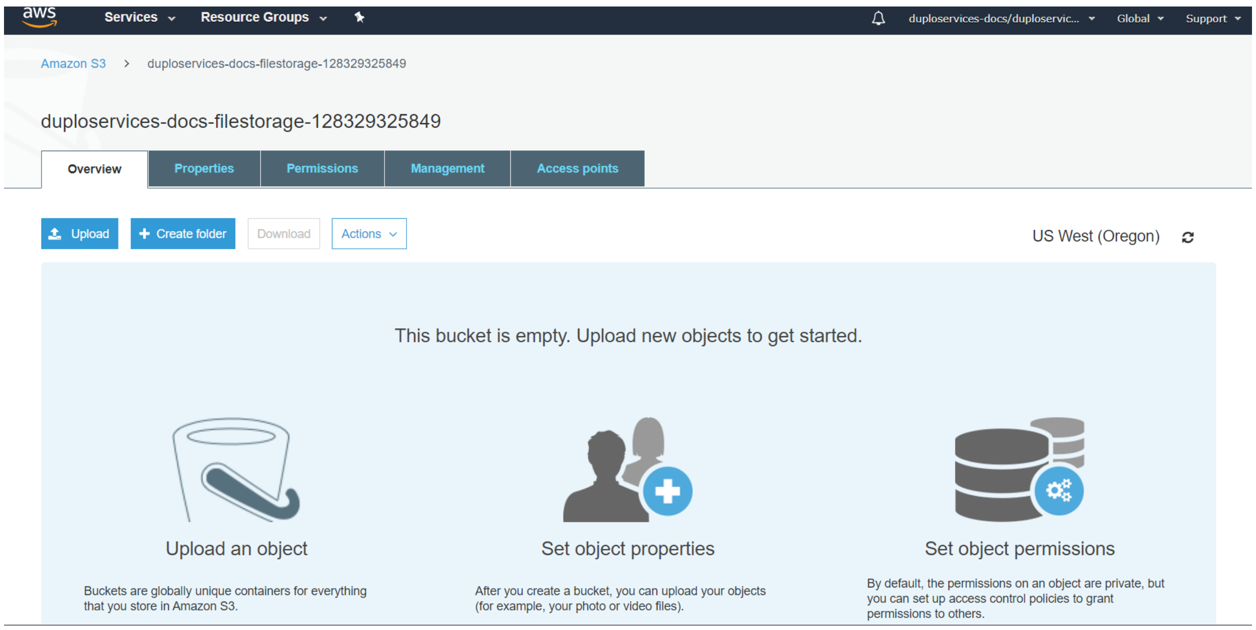
Public Access: false
Default Encryption: Sse
Versioning: false
Access Logs: false
Policies

Total 0 Show 10

KEY	VALUE
No data to display	

0 total

- Now click on the AWS Console button under Actions menu, this will take us directly to the S3 bucket in AWS console with limited access. Here, we can upload the file and modify permissions. Select Upload and choose the text file that we created earlier. Hit next till the end and the file should be uploaded.



This bucket is empty. Upload new objects to get started.

Upload an object

Buckets are globally unique containers for everything that you store in Amazon S3.

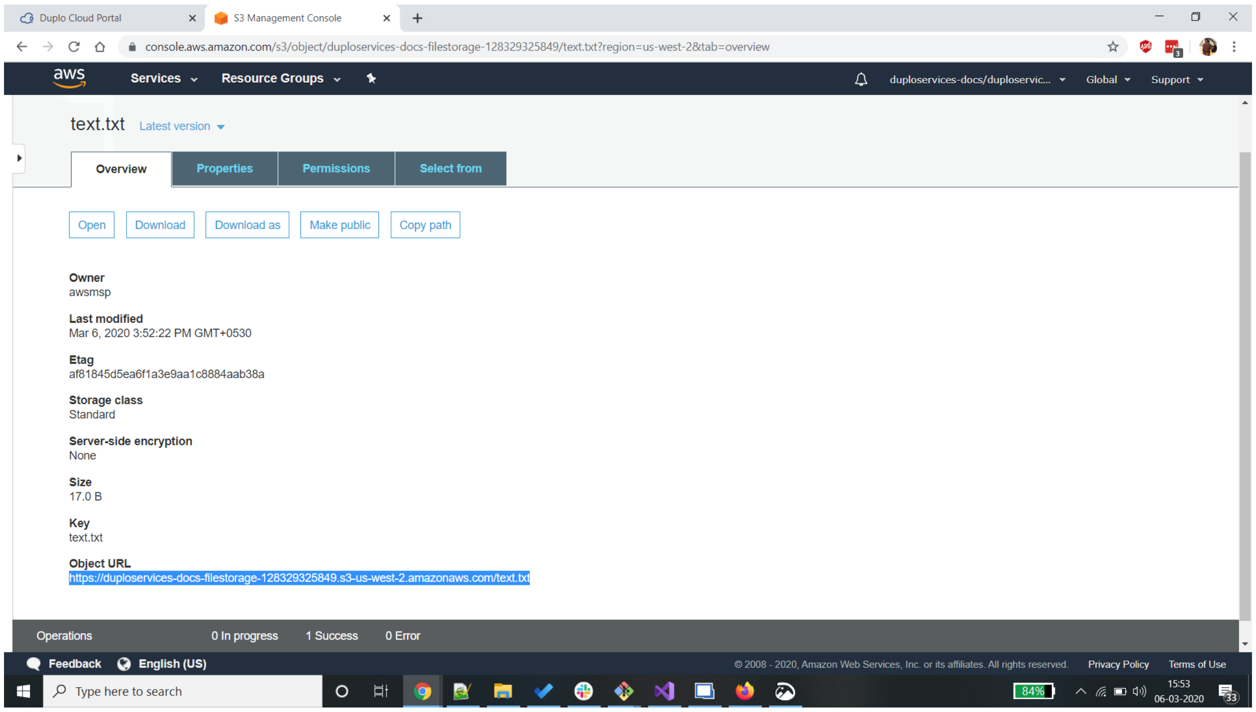
Set object properties

After you create a bucket, you can upload your objects (for example, your photo or video files).

Set object permissions

By default, the permissions on an object are private, but you can set up access control policies to grant permissions to others.

- Click on the file name and copy the Link for the file in the bucket.



Owner
awsmsp

Last modified
Mar 6, 2020 3:52:22 PM GMT+0530

Etag
af81845d5ea6f1a3e9aa1c8884aab38a

Storage class
Standard

Server-side encryption
None

Size
17.0 B

Key
text.txt

Object URL
https://duploservices-docs-filestorage-128329325849.s3-us-west-2.amazonaws.com/text.txt

- Now let's assign this link to the link variable in app.js

```javascript
var express = require('express')
var app = express()
var request = require('request');
var message = '';
var link = 'https://s3-us-west-2.amazonaws.com/duploservices-670rp5-ac42540e-1401-49ee-939e-0c0e8c474263/myfile.txt';

request(link, function (error, response, body) {
  console.log('error:', error); // Print the error if one occurred
  console.log('statusCode:', response && response.statusCode); // Print the response status code if a response was received
  console.log('body:', body); // Print the HTML for the Google homepage.
  message = body;
});

app.get('/', function (req, res) {
  res.send('Hello '+ message)
})

app.listen(3000, function () {
  console.log('Example app listening on port 3000!')
})
```

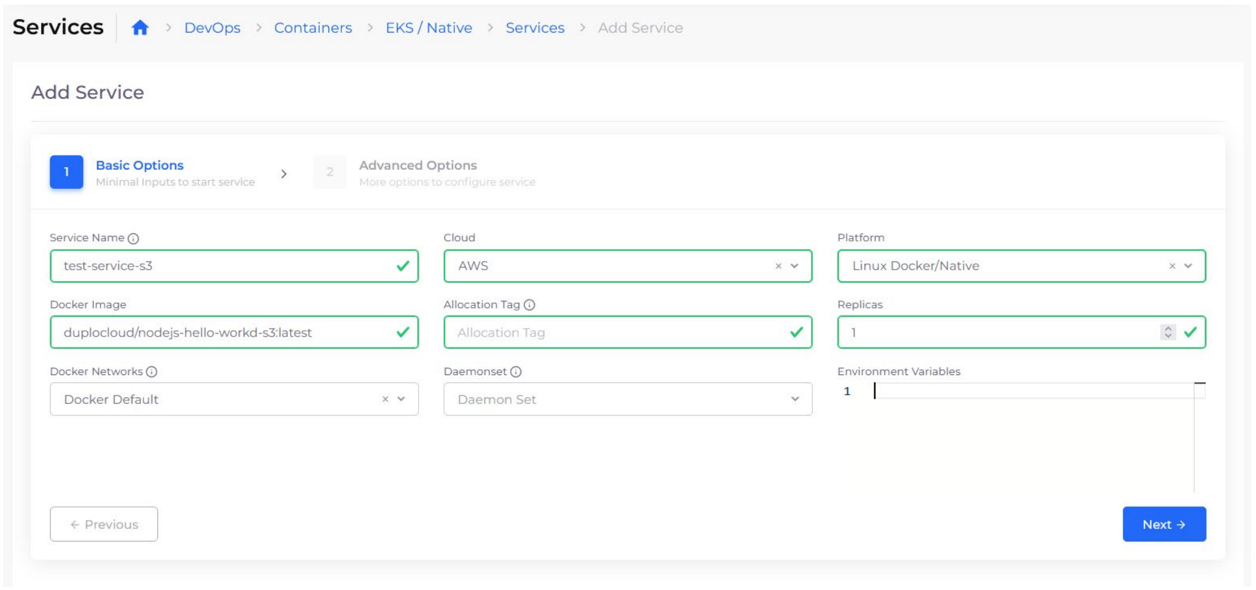
- Almost there, all we need to do now is create a Docker file and push to Docker Hub so that DuploCloud can access it.
- I have created a simple dockerfile inspired from this Blog from NodeJS.

```dockerfile
FROM node:boron

# Create app directory
RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app

# Install app dependencies
COPY package.json /usr/src/app/
RUN npm install

# Bundle app source
COPY . /usr/src/app

EXPOSE 3000
CMD [ "nodejs", "app.js" ]
```

- We will now run the Docker build command to build out an image from this dockerfile.

- And now Docker push to push it to Docker Hub

- All set! We will create a new service as we did in the last tutorial to run this new Docker container.
- In the services tab click on the plus sign and fill the options as specified.

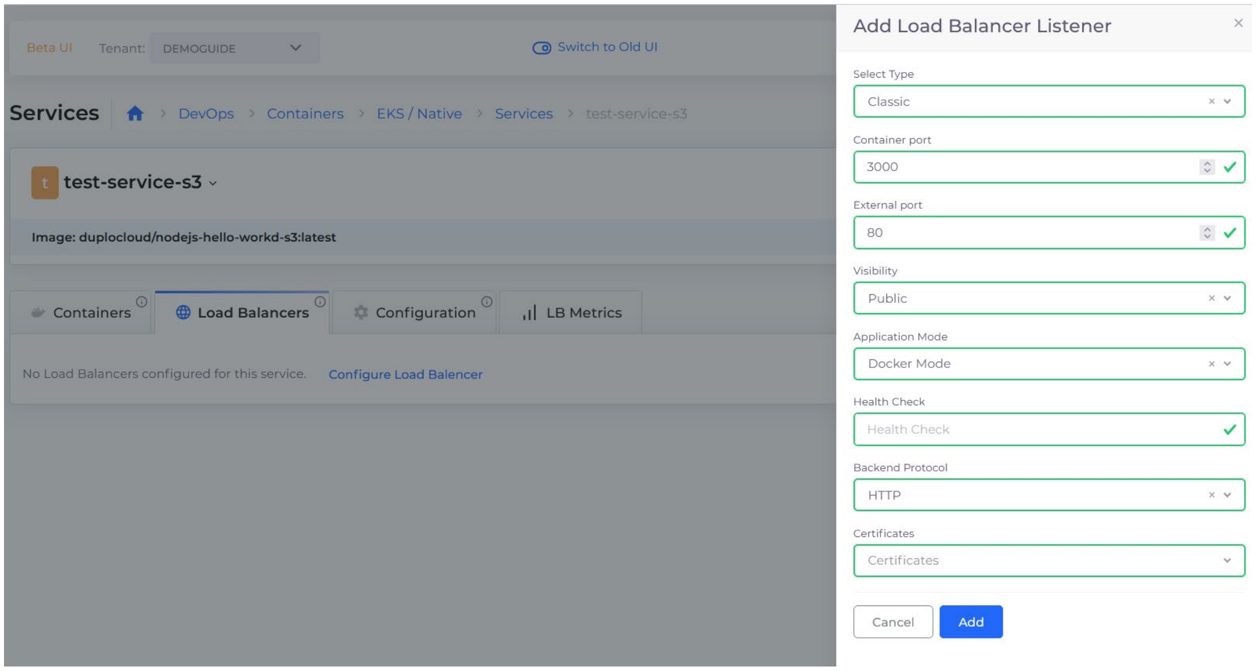

Services > DevOps > Containers > EKS / Native > Services > Add Service

Add Service

1 Basic Options — Minimal Inputs to start service
2 Advanced Options — More options to configure service

Service Name: test-service-s3	Cloud: AWS	Platform: Linux Docker/Native
Docker Image: duplocloud/nodejs-hello-workd-s3:latest	Allocation Tag: Allocation Tag	Replicas: 1
Docker Networks: Docker Default	Daemonset: Daemon Set	Environment Variables: 1

- It's important to note here that this service along with the previous service will run in the single host that we created in the previous tutorial! All we need to do is create a new load balancer for this service so that we can access it.
- Click on configure Load balancer button under Load Balancer tab of the service and fill out the form.


Beta UI Tenant: DEMOGUIDE Switch to Old UI

Services > DevOps > Containers > EKS / Native > Services > test-service-s3

test-service-s3

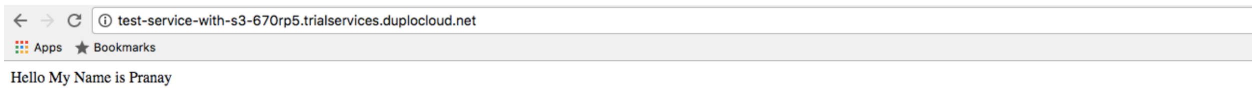
Image: duplocloud/nodejs-hello-workd-s3:latest

No Load Balancers configured for this service. Configure Load Balencer

Add Load Balancer Listener

Select Type	Classic
Container port	3000
External port	80
Visibility	Public
Application Mode	Docker Mode
Health Check	Health Check
Backend Protocol	HTTP
Certificates	Certificates

- As before, give it a minute and hit the freshly generated link in the browser. Voila!


test-service-with-s3-670rp5.trialservices.duplocloud.net

Hello My Name is Pranay

- So now we have two services which we can access via two different load balancers running on a single host (EC2 instance)! Due to this tight coupling with S3 we can do all sorts of fun stuff like store static files as well as javascript in S3.

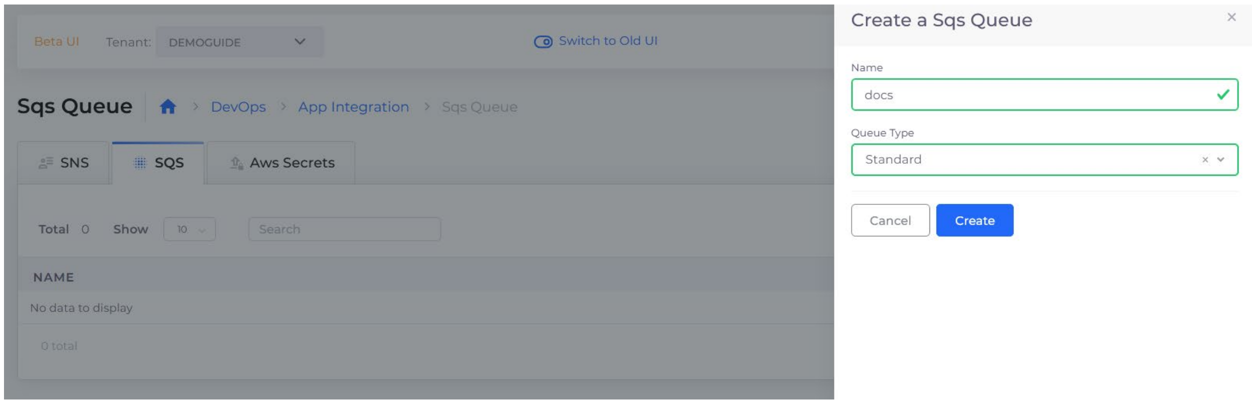
Serverless Webapp

DuploCloud makes deploying serverless applications a breeze. A step by step tutorial on how to deploy a Lambda application is at https://www.linkedin.com/pulse/deploy-existing-web-applications-serverless-using-aws-thiruvengadam/

Following is a video version of the tutorial: https://youtu.be/MW62CrzVO2E

Deployment is a three-step process.

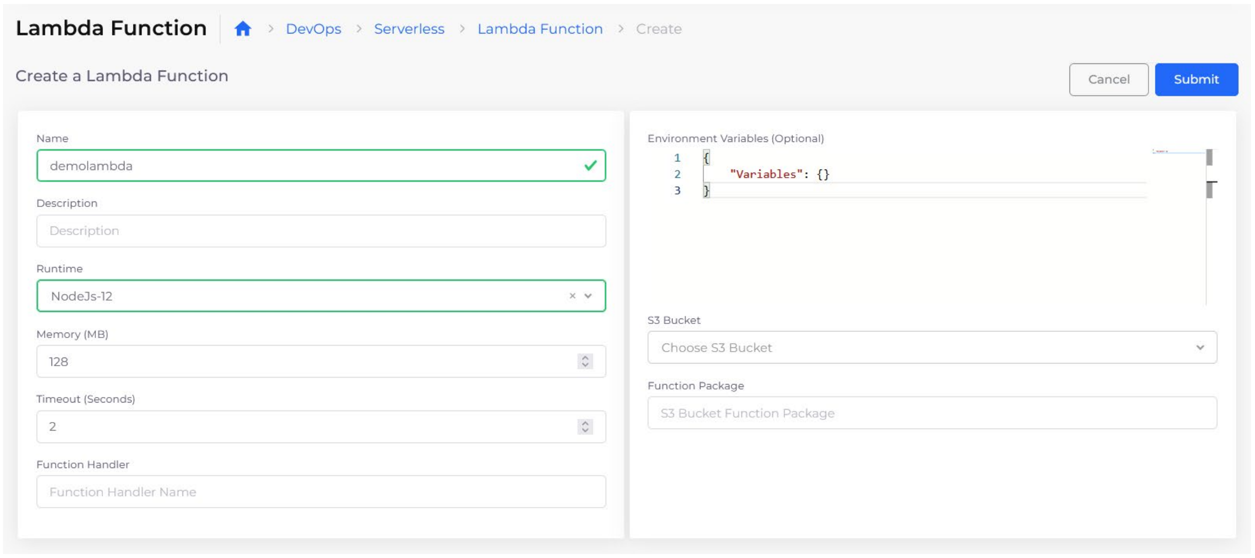
- **Create a Zip file:** Generate a Zip package of your Lambda code. The Lambda function code should be at the root of the package. If you are using virtual env, all dependencies should be packaged. Refer to the AWS documentation at http://docs.aws.amazon.com/lambda/latest/dg/lambda-python-how-to-create-deployment-package.html for detailed instructions on how to generate a package. We personally prefer using tools like zappa and serverless.
- **Create an S3 bucket and upload the Zip file: DevOps → Storage → S3 → +Add** button above the table. Give a name for your bucket or leave it blank and a name will be auto generated. Then Click on AWS Console Button to get into AWS console for this S3 bucket and upload the zip file we just created.



Name: docs

Queue Type: Standard

- **Create Lambda Function: DevOps → Serverless → Lambda → +Add** button above the table. Give a name for the Lambda function and other values. This will create the lambda function. Click on AWS console to go to the AWS console for this function. Test the function. Refer to Lambda for more info. You can also look at the tutorial above for the same or look at AWS documentation.



Name: demolambda
Description: Description
Runtime: NodeJs-12
Memory (MB): 128
Timeout (Seconds): 2
Function Handler: Function Handler Name
Environment Variables (Optional):

```
{
    "Variables": {}
}
```

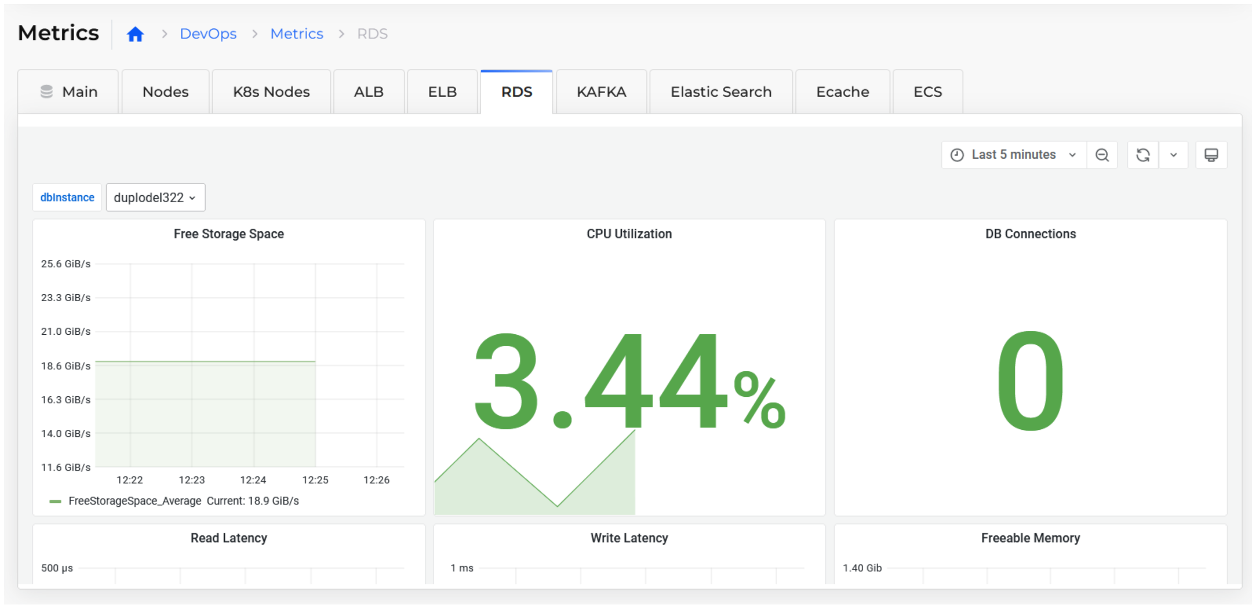
S3 Bucket: Choose S3 Bucket
Function Package: S3 Bucket Function Package

Passing Secrets

Passing secrets to the Docker based service/ Lambda function can be done using environmental variables. For example, you can create a SQL database from the **DevOps → Database → RDS** tab in DuploCloud, provide username and password, then in the Lambda menu give the same username and password. No secrets need to be stored anywhere outside like vault, git, repo, etc.

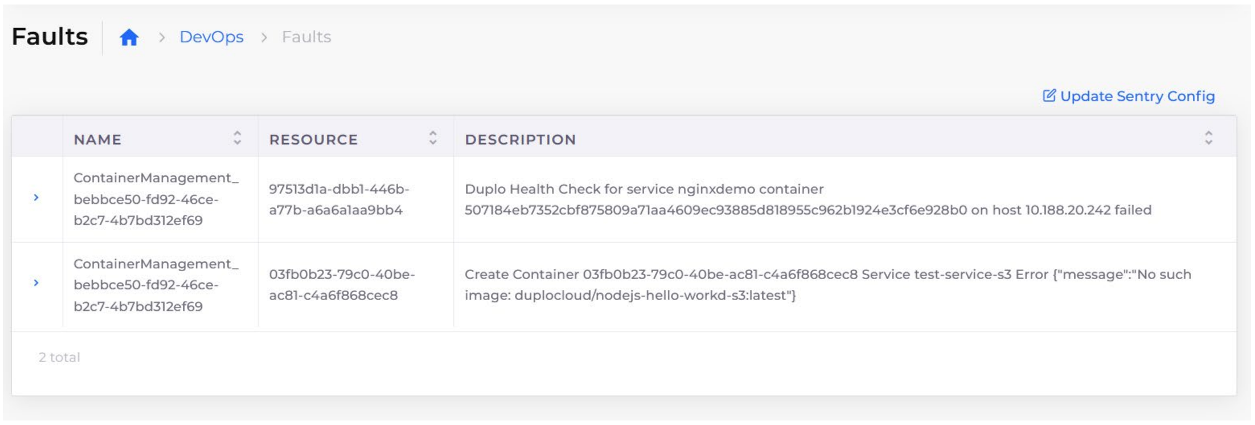
Monitoring

Metrics of the resources created/managed in DuploCloud can be tracked under **DevOps → Metrics**.



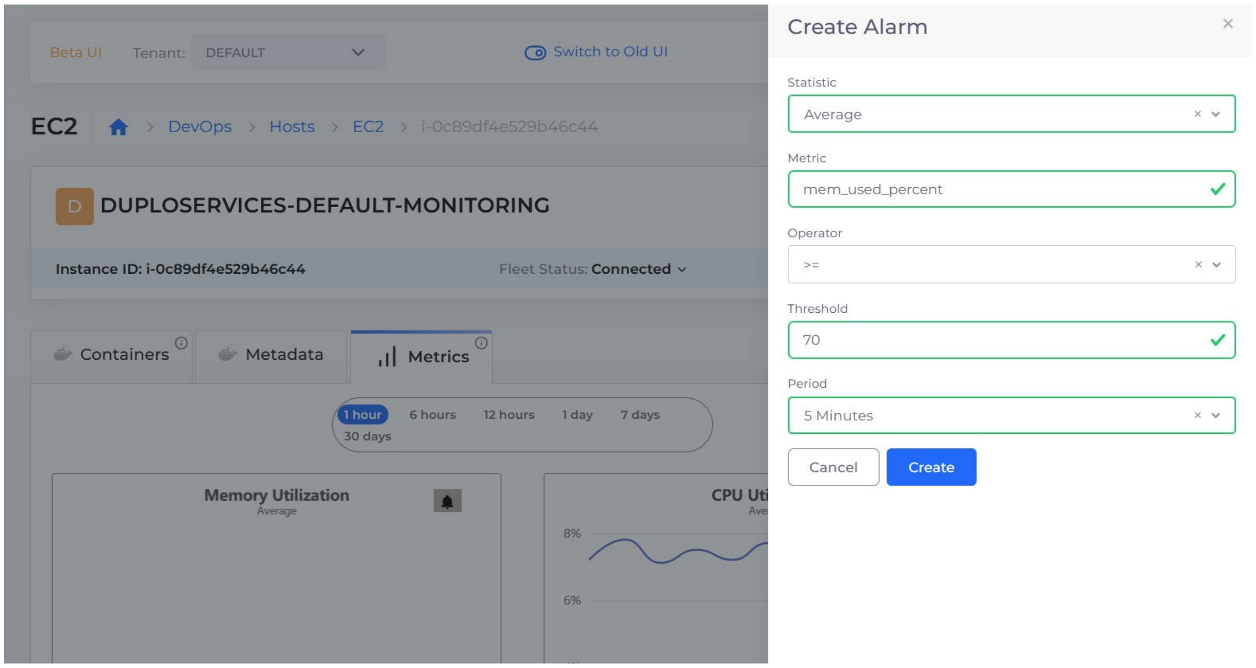
Faults/Alarms

Faults that happen in the system, be it Infrastructure creation, container deployments, Application health checks, or any Triggered Alarms can be tracked in the DuploCloud portal under Faults Menu. You can look at Tenant-specific faults under **DevOps → Faults** or all the faults in the system under **Administrator → Faults**. You can set the AWS Alarms for the individual metrics, click on the bell icon on any of the metrics. A form to create an alarm shows up. You can provide the necessary information and create the alarm.



NAME	RESOURCE	DESCRIPTION
ContainerManagement_bebbce50-fd92-46ce-b2c7-4b7bd312ef69	97513d1a-dbb1-446b-a77b-a6a6a1aa9bb4	Duplo Health Check for service nginxdemo container 507184eb7352cbf875809a71aa4609ec93885d818955c962b1924e3cf6e928b0 on host 10.188.20.242 failed
ContainerManagement_bebbce50-fd92-46ce-b2c7-4b7bd312ef69	03fb0b23-79c0-40be-ac81-c4a6f868cec8	Create Container 03fb0b23-79c0-40be-ac81-c4a6f868cec8 Service test-service-s3 Error {"message":"No such image: duplocloud/nodejs-hello-workd-s3:latest"}

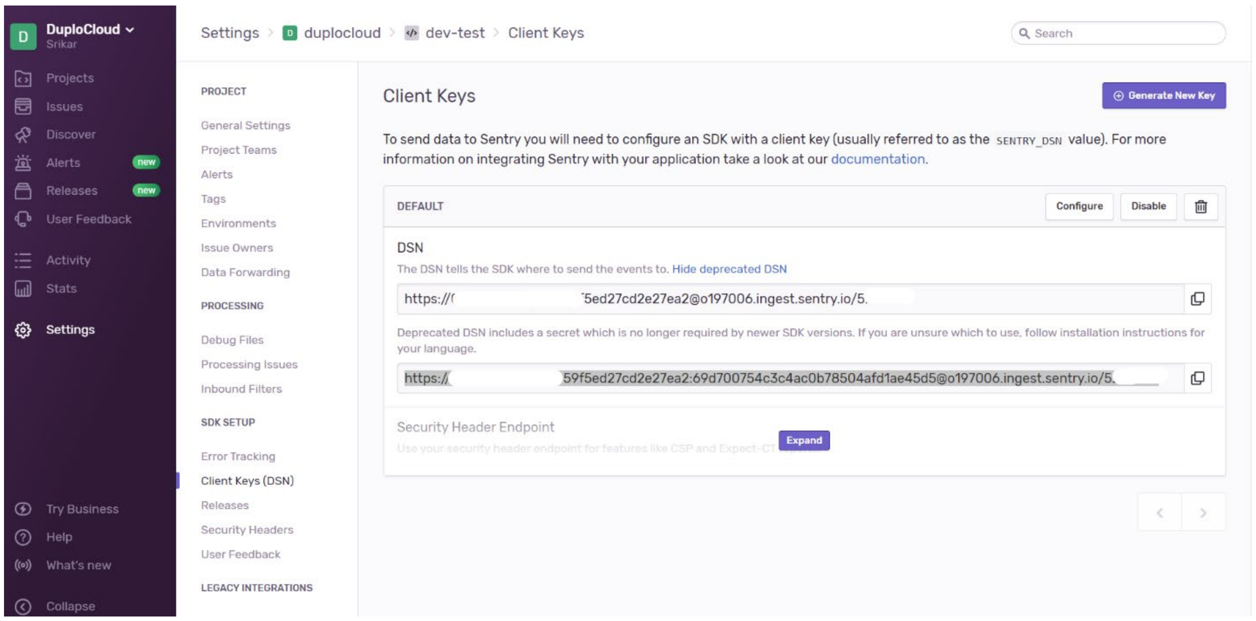
2 total



Statistic: Average
Metric: mem_used_percent
Operator: >=
Threshold: 70
Period: 5 Minutes

Alerting/Notification

In addition to notify you about the faults, DuploCloud integrates with Sentry, which will send an Email alert for the fault and acts as a single place to look at all the events. To configure Sentry, go to https://sentry.io/ Under projects create a new project. Then go to **Settings → Projects → your project → Client keys (DSN)**. Click on show deprecated DSN, Get the deprecated DNS name and add it under **DevOps → Faults → Update Sentry Config**.



Client Keys

To send data to Sentry you will need to configure an SDK with a client key (usually referred to as the `SENTRY_DSN` value). For more information on integrating Sentry with your application take a look at our documentation.

DEFAULT

DSN

The DSN tells the SDK where to send the events to. Hide deprecated DSN

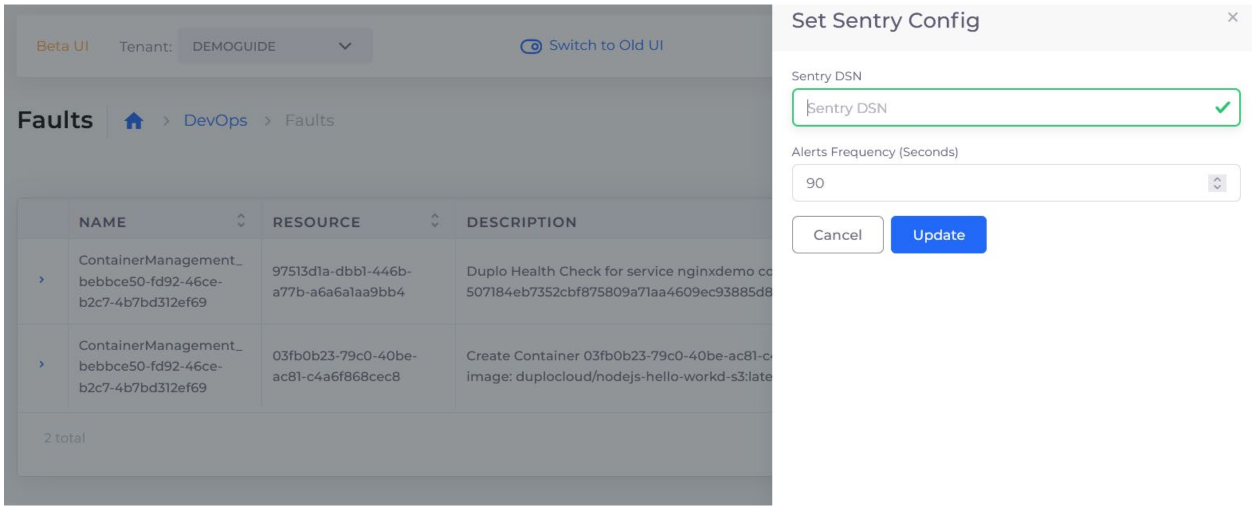
https://(5ed27cd2e27ea2@o197006.ingest.sentry.io/5.

Deprecated DSN includes a secret which is no longer required by newer SDK versions. If you are unsure which to use, follow installation instructions for your language.

https:/(59f5ed27cd2e27ea2:69d700754c3c4ac0b78504afd1ae45d5@o197006.ingest.sentry.io/5.

Security Header Endpoint

Use your security header endpoint for features like CSP and Expect-CT



	NAME	RESOURCE	DESCRIPTION
›	ContainerManagement_bebbce50-fd92-46ce-b2c7-4b7bd312ef69	97513d1a-dbb1-446b-a77b-a6a6a1aa9bb4	Duplo Health Check for service nginxdemo co 507184eb7352cbf875809a71aa4609ec93885d8
›	ContainerManagement_bebbce50-fd92-46ce-b2c7-4b7bd312ef69	03fb0b23-79c0-40be-ac81-c4a6f868cec8	Create Container 03fb0b23-79c0-40be-ac81-c image: duplocloud/nodejs-hello-workd-s3:late

2 total

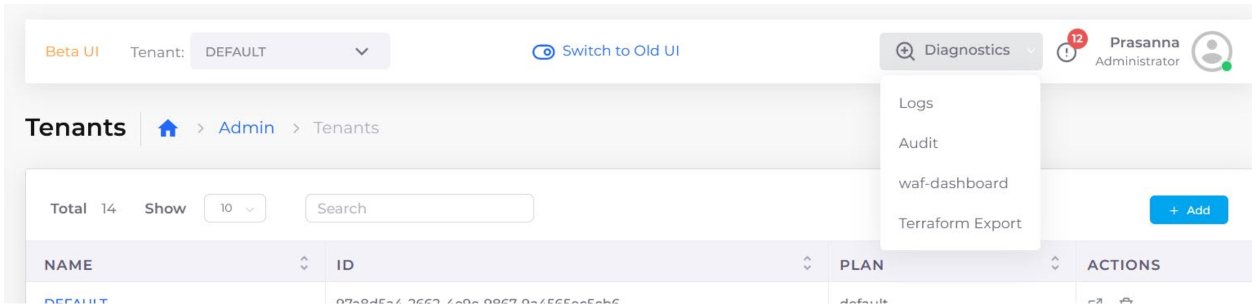
Set Sentry Config

Sentry DSN: Sentry DSN

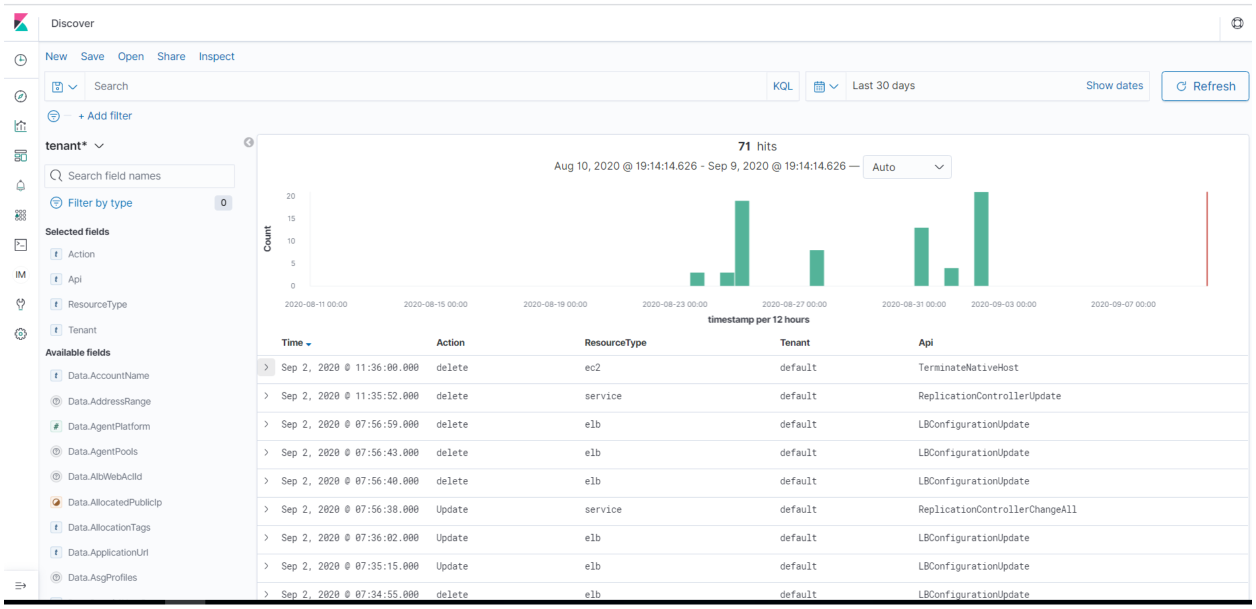
Alerts Frequency (Seconds): 90

Central Logging

All the activity in the DuploCloud is logged which can be used for auditing. All the logs are saved into Elasticsearch and can be visualized in Kibana. The URL for the Kibana is available under Diagnostics.



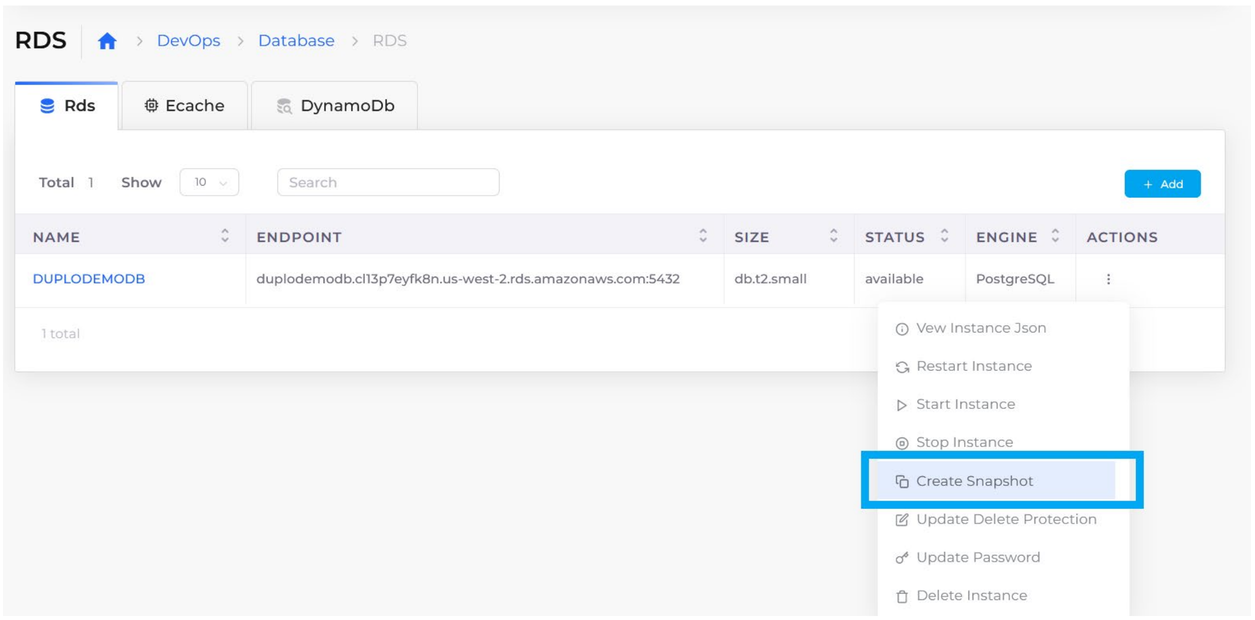
The ES & Kibana will be sitting inside the VPC and cannot be accessed from outside. Connect to the VPN and access these URL.



Time	Action	ResourceType	Tenant	Api
Sep 2, 2020 @ 11:36:00.000	delete	ec2	default	TerminateNativeHost
Sep 2, 2020 @ 11:35:52.000	delete	service	default	ReplicationControllerUpdate
Sep 2, 2020 @ 07:56:59.000	delete	elb	default	LBConfigurationUpdate
Sep 2, 2020 @ 07:56:43.000	delete	elb	default	LBConfigurationUpdate
Sep 2, 2020 @ 07:56:40.000	delete	elb	default	LBConfigurationUpdate
Sep 2, 2020 @ 07:56:38.000	Update	service	default	ReplicationControllerChangeAll
Sep 2, 2020 @ 07:36:02.000	Update	elb	default	LBConfigurationUpdate
Sep 2, 2020 @ 07:35:15.000	Update	elb	default	LBConfigurationUpdate
Sep 2, 2020 @ 07:34:55.000	delete	elb	default	LBConfigurationUpdate

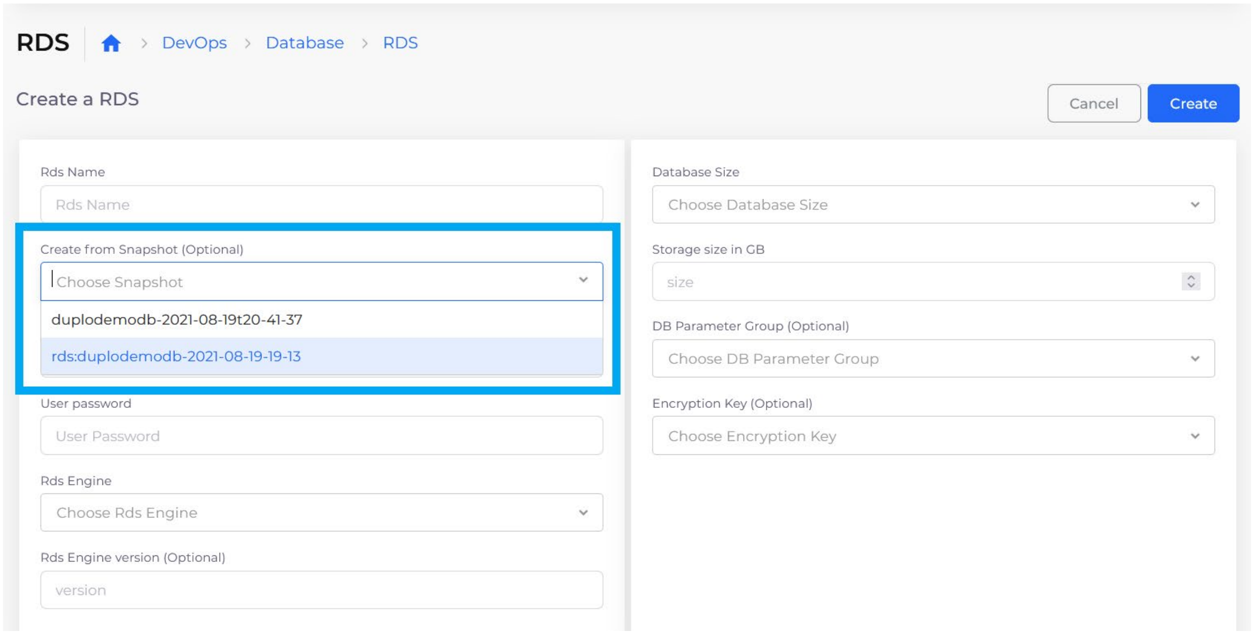
Backup & Restore

Backup of the resources, be it DB Snaphsots, VM Snaphsots/VM AMI can be done directly under the resource in the DuploCloud. Once the backups are done, they will be available to be restored on the next instance creation.



NAME	ENDPOINT	SIZE	STATUS	ENGINE	ACTIONS
DUPLODEMODB	duplodemodb.cl13p7eyfk8n.us-west-2.rds.amazonaws.com:5432	db.t2.small	available	PostgreSQL	⋮

1 total

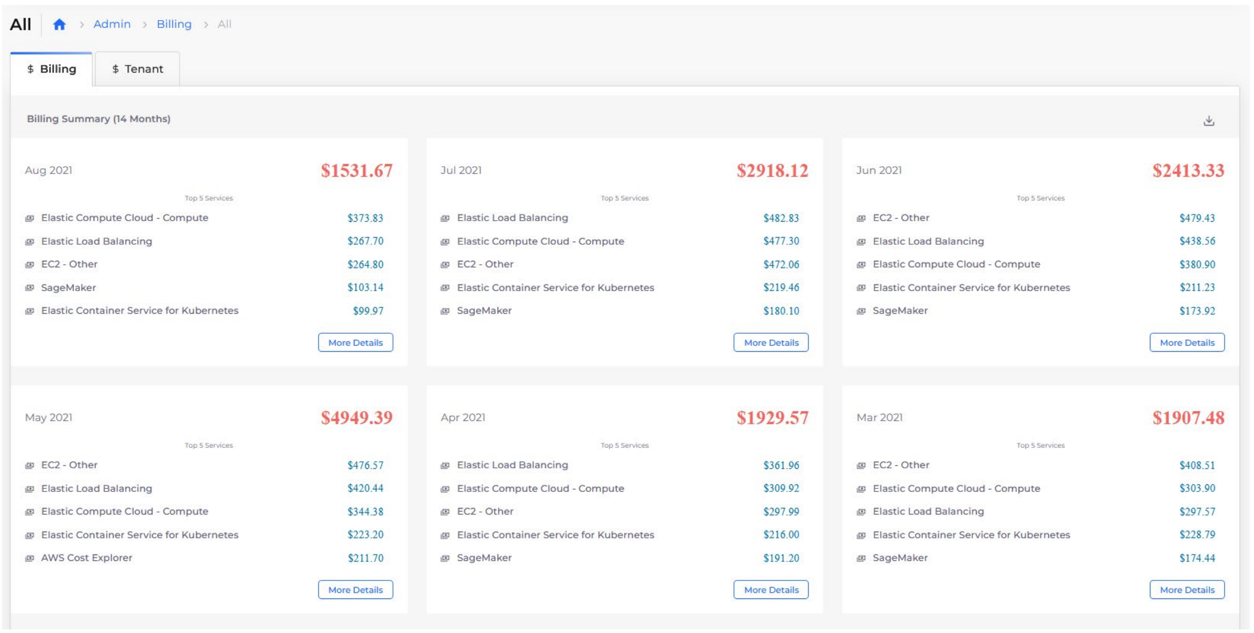


Disaster Recovery

Documentation TBD. Please contact DuploCloud team for assistance.

Cost Management

Usage cost for the resources can be viewed in the DuploCloud under **Administrator → Billing**. You can explore the cost for any period and for any set of resources.


All > Admin > Billing > All

$ Billing | $ Tenant

Billing Summary (14 Months)

Aug 2021	$1531.67
Top 5 Services	
Elastic Compute Cloud - Compute	$373.83
Elastic Load Balancing	$267.70
EC2 - Other	$264.80
SageMaker	$103.14
Elastic Container Service for Kubernetes	$99.97

Jul 2021	$2918.12
Top 5 Services	
Elastic Load Balancing	$482.83
Elastic Compute Cloud - Compute	$477.30
EC2 - Other	$472.06
Elastic Container Service for Kubernetes	$219.46
SageMaker	$180.10

Jun 2021	$2413.33
Top 5 Services	
EC2 - Other	$479.43
Elastic Load Balancing	$438.56
Elastic Compute Cloud - Compute	$380.90
Elastic Container Service for Kubernetes	$211.23
SageMaker	$173.92

May 2021	$4949.39
Top 5 Services	
EC2 - Other	$476.57
Elastic Load Balancing	$420.44
Elastic Compute Cloud - Compute	$344.38
Elastic Container Service for Kubernetes	$223.20
AWS Cost Explorer	$211.70

Apr 2021	$1929.57
Top 5 Services	
Elastic Load Balancing	$361.96
Elastic Compute Cloud - Compute	$309.92
EC2 - Other	$297.99
Elastic Container Service for Kubernetes	$216.00
SageMaker	$191.20

Mar 2021	$1907.48
Top 5 Services	
EC2 - Other	$408.51
Elastic Compute Cloud - Compute	$303.90
Elastic Load Balancing	$297.57
Elastic Container Service for Kubernetes	$228.79
SageMaker	$174.44

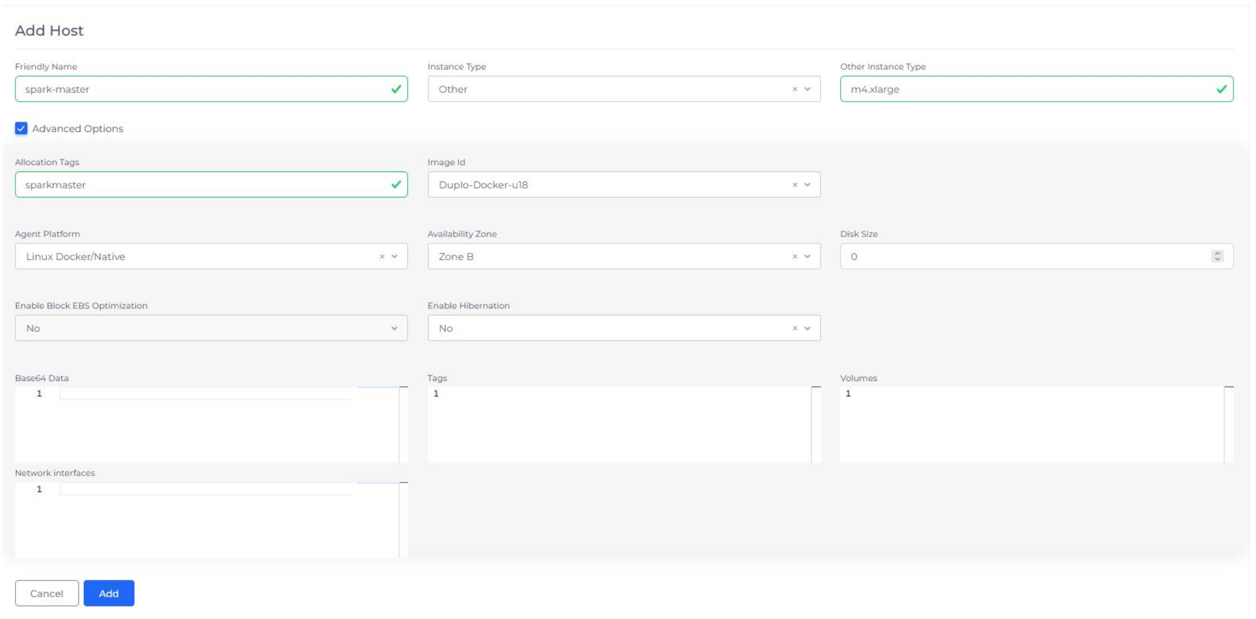
Bigdata/ETL

- **Use case:**
  - **Collection of data from using various methods/sources**
    - Web scraping: Selenium using headless chrome/firefox.
    - Web crowling: statis website sing crowling
    - API to Data collection: It could be REST or GraphQL API
    - Private internal customer data collected over various transactions
    - Private external customer data collected over secured SFTP
    - The data purchased from 3rd party
    - The data from various social networks
  - **Correlate data from various sources**
    - Clean up and Process data and apply various statistical methods, create
    - Correlate terabytes of data from various sources and make sense from the data.

        - Detect anomalies, summarize, bucketize, and various aggregations
        - Attach meta-data to enrich data.
        - Create data for NLP and ML models for predictions of future events.
    - **AI/ML pipelines and life-cycle management**
        - Make data available to data science team
        - Train models and do continues improvement trials, reinforcement learning.
        - Create anomalies, bucketize data, summarize and do various aggragations.
        - Train NLP and ML models for predictions of future events based on history
        - Create history for models/hyper parameters and data at various stages.
- **Deploying Apache Spark™ cluster**

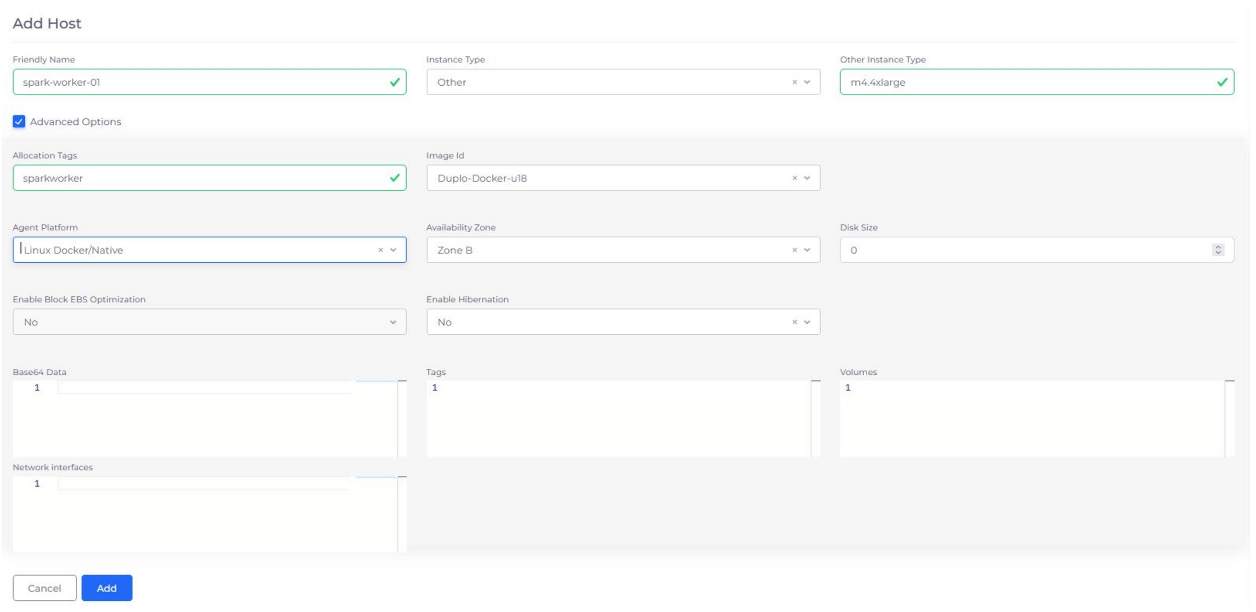
  In this tutorial we will create a Spark cluster with a Jupyter notebook. A typical use case is ETL jobs, for example reading parquet files from S3, processing and pushing reports to databases. The aim is to process GBs of data in faster and cost-effective way. The high-level steps are: (1) Create 3 VMs one for each Spark master, Spark worker and Jupyter notebook. (2) Deploy Docker images for each of these on these VMs.

    - **Create a host for Spark master:** Click **DevOps → Hosts → EC2 → +Add** button to create hosts and click **show advanced**. Change the value of instance type to '`m4.xlarge`' and add an allocation tag '`sparkmaster`'.


Add Host

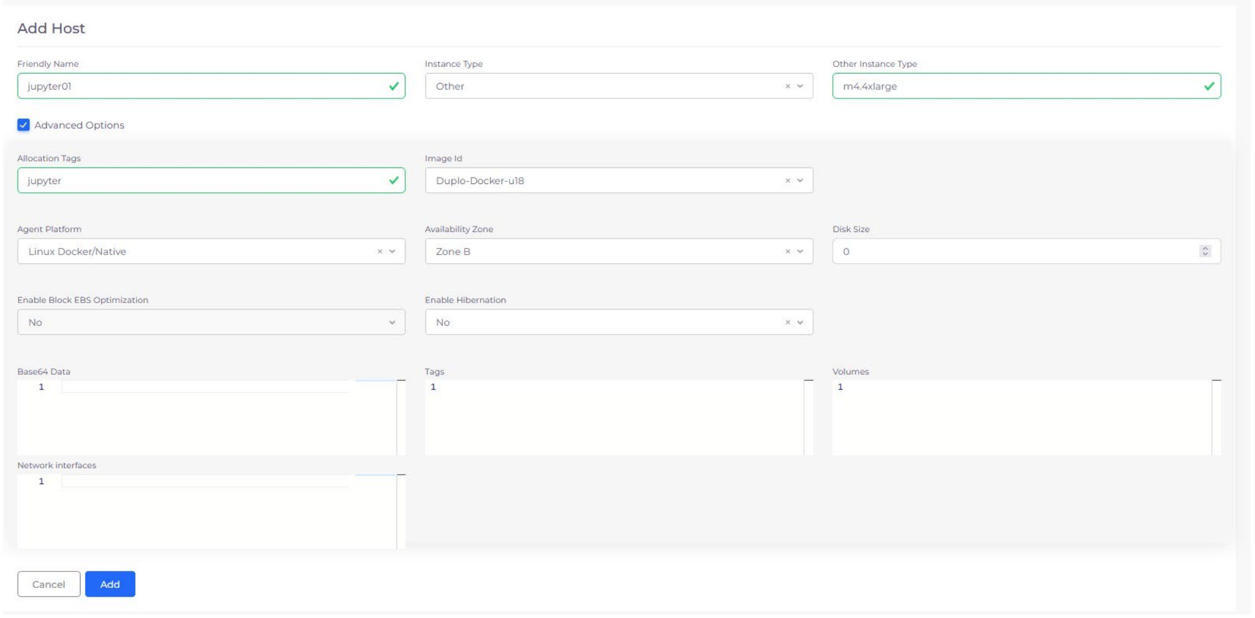
Friendly Name: spark-master
Instance Type: Other
Other Instance Type: m4.xlarge
Advanced Options ✓
Allocation Tags: sparkmaster
Image Id: Duplo-Docker-u18
Agent Platform: Linux Docker/Native
Availability Zone: Zone B
Disk Size: 0
Enable Block EBS Optimization: No
Enable Hibernation: No
Base64 Data: 1
Tags: 1
Volumes: 1
Network interfaces: 1

- **Create host for spark worker:** Create another host for the worker. Change the value of instance type to '`m4.4xlarge`' and add an allocation tag '`sparkworker`'. Click on submit. The number of workers depends on how much load you want to process. You should add one host for each worker. They should all have the same allocation tag '`sparkworker`'. You can add/remove workers and scale up or down the worker Spark service as many times as you want. We will see in the next steps:


Add Host

Friendly Name: spark-worker-01
Instance Type: Other
Other Instance Type: m4.4xlarge
Advanced Options ✓
Allocation Tags: sparkworker
Image Id: Duplo-Docker-u18
Agent Platform: Linux Docker/Native
Availability Zone: Zone B
Disk Size: 0
Enable Block EBS Optimization: No
Enable Hibernation: No
Base64 Data: 1
Tags: 1
Volumes: 1
Network interfaces: 1

- **Create host for jupyter:** Create one more host for Jupyter notebook. Choose the value of instance type to '`m4.4xlarge`' and add the allocation tag as '`jupyter`'.

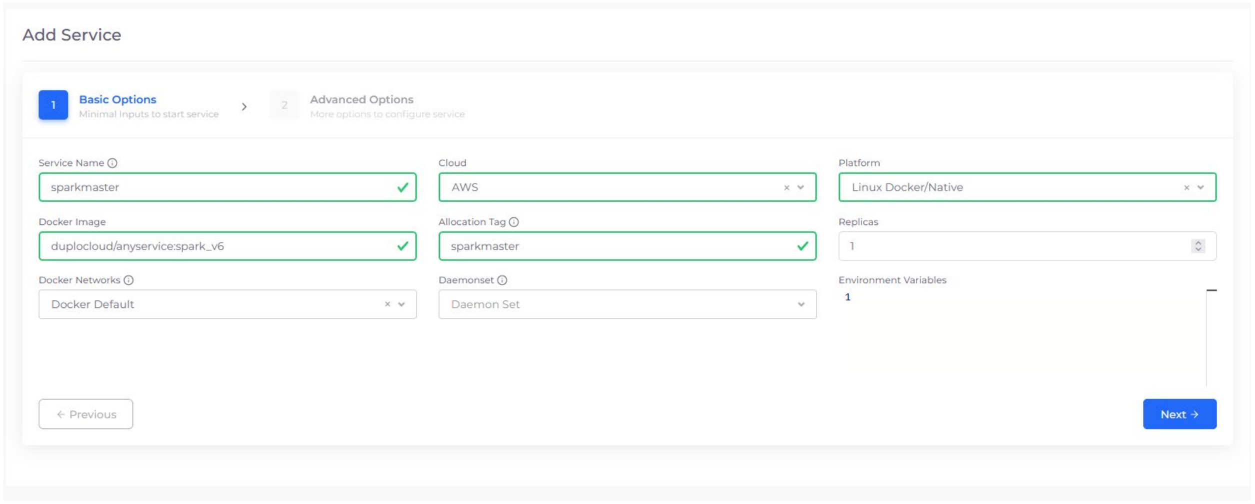

Add Host

Friendly Name: jupyter01
Instance Type: Other
Other Instance Type: m4.4xlarge

☑ Advanced Options

Allocation Tags: jupyter
Image Id: Duplo-Docker-u18
Agent Platform: Linux Docker/Native
Availability Zone: Zone B
Disk Size: 0
Enable Block EBS Optimization: No
Enable Hibernation: No
Base64 Data
Tags
Volumes
Network interfaces

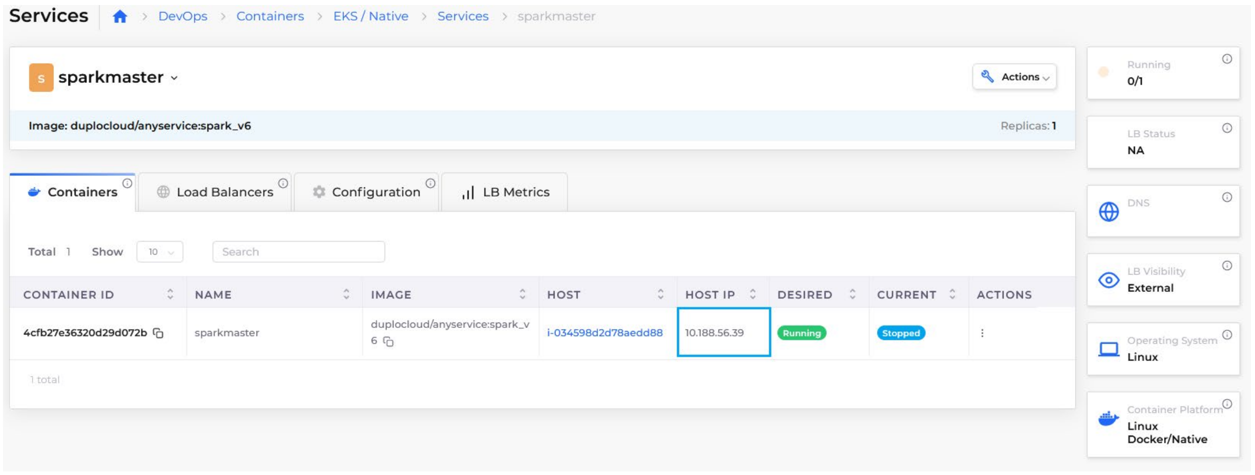
- **Create spark master Docker service:** Go to **Containers → EKS/ Native → Services** and create new service. Under name choose '`sparkmaster`', image '`duplocloud/anyservice:spark_v6`', add the allocation tag '`sparkmaster`'. In the Docker host config select Host network. By setting this in Docker Host config you are making the container networking the same as the VM i.e., container IP is same as VM.


Add Service

1 Basic Options
Minimal inputs to start service
2 Advanced Options
More options to configure service

Service Name: sparkmaster
Cloud: AWS
Platform: Linux Docker/Native
Docker Image: duplocloud/anyservice:spark_v6
Allocation Tag: sparkmaster
Replicas: 1
Docker Networks: Docker Default
Daemonset: Daemon Set
Environment Variables

- **Create Jupyter service:** First we need the IP address of Spark master. Click on Spark master service and on the right expand the container details and copy the host IP. Create another service, under name choose '`jupyter`', image '`duplocloud/anyservice:spark_notebook_pyspark_scala_v4`', add the allocation jupyter and select Host network for Docker Host Config, Add volume mapping "`/home/ubuntu/jupyter:/home/jovyan/work`", Also provide the environment variables {"`SPARK_OPTS":" --master spark://<>:7077 --driver-memory 20g --executor-memory 15g --executor-cores 4 `"}, replace <> with the IP you just got above.



Services › DevOps › Containers › EKS / Native › Services › sparkmaster

sparkmaster

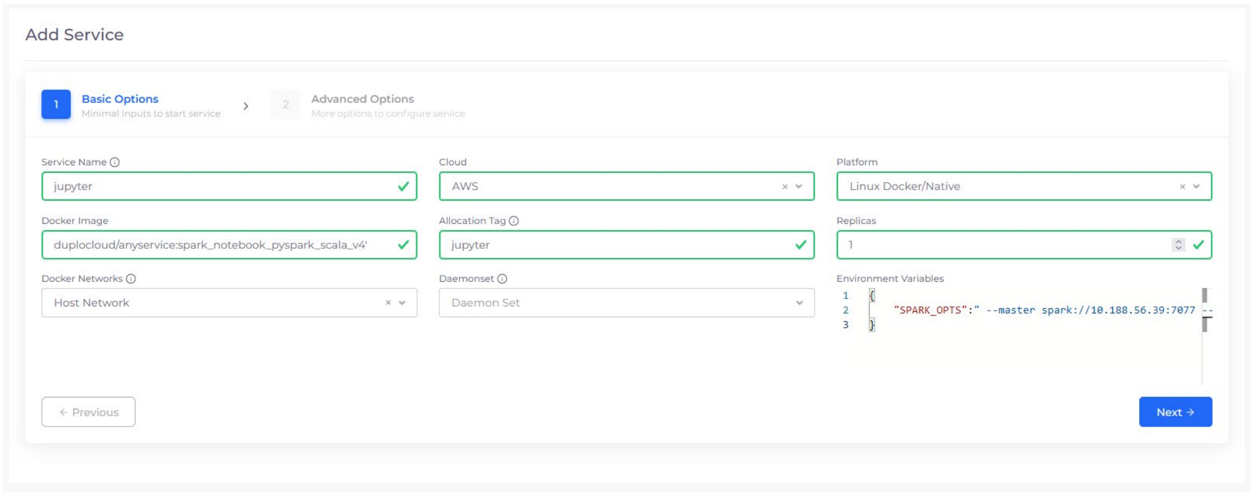
Image: duplocloud/anyservice:spark_v6 — Replicas: 1

Running 0/1 · LB Status NA · DNS · LB Visibility External · Operating System Linux · Container Platform Linux Docker/Native

Total 1 Show 10

CONTAINER ID	NAME	IMAGE	HOST	HOST IP	DESIRED	CURRENT	ACTIONS
4cfb27e36320d29d072b	sparkmaster	duplocloud/anyservice:spark_v6	i-034598d2d78aedd88	10.188.56.39	Running	Stopped	⋮

1 total



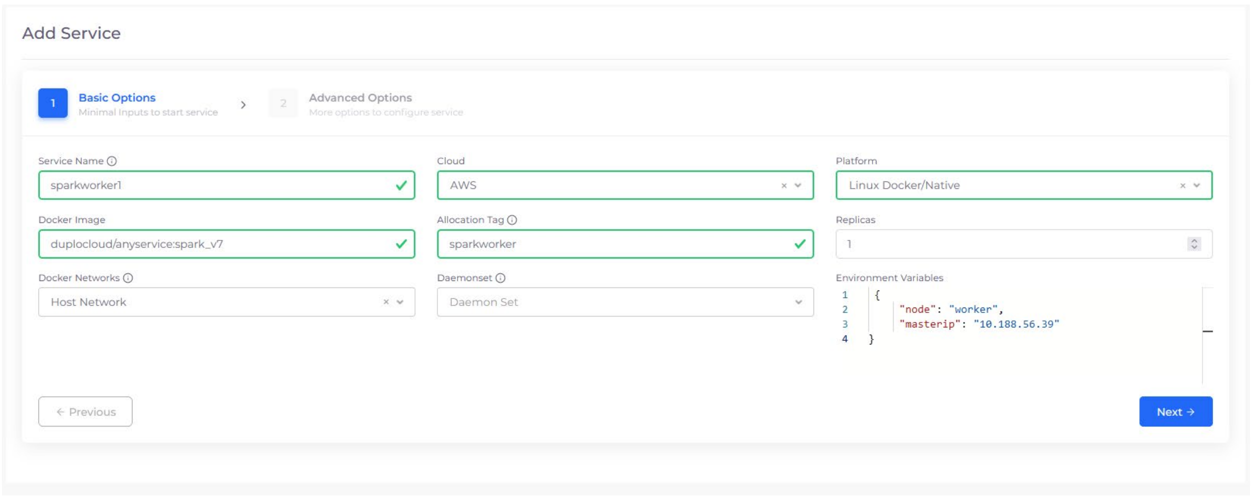
Add Service

1 Basic Options — Minimal inputs to start service › 2 Advanced Options — More options to configure service

Service Name: jupyter	Cloud: AWS	Platform: Linux Docker/Native
Docker Image: duplocloud/anyservice:spark_notebook_pyspark_scala_v4'	Allocation Tag: jupyter	Replicas: 1
Docker Networks: Host Network	Daemonset: Daemon Set	Environment Variables

```
{
    "SPARK_OPTS":" --master spark://10.188.56.39:7077 --
}
```

- **Create Spark workers:** Create another service name '`sparkworker`', image '`duplocloud/anyservice:spark_v7`', add the allocation sparkworker and select Host network for Docker Host Config. Also provide the environment varibales { `"node": "worker", "masterip": "<"`}, replace <> with the IP you just got above. Depending on how many worker hosts you have created, use the same number under replicas and that is the way you can scale up and down. At any time, you can add new hosts, set the allocation tag sparkworker and then under services, edit the sparkworker service and update the replicas.


Add Service

1 Basic Options
Minimal inputs to start service
2 Advanced Options
More options to configure service

Service Name: sparkworker1	Cloud: AWS	Platform: Linux Docker/Native
Docker Image: duplocloud/anyservice:spark_v7	Allocation Tag: sparkworker	Replicas: 1
Docker Networks: Host Network	Daemonset: Daemon Set	Environment Variables

Environment Variables:

```
{
    "node": "worker",
    "masterip": "10.188.56.39"
}
```

- **Create Docker services shell :** Add/update shell access by clicking on >_ icon. This gives you easy access into the container shell. You will need to wait for 5 minutes for the shell to be ready. Make sure you are connected to VPN if you choose to launch the shell as internal only
- **Open Jupyter docker shell:** Select Jupyter service and expand the container. Copy the hostip and then click on >_ icon.

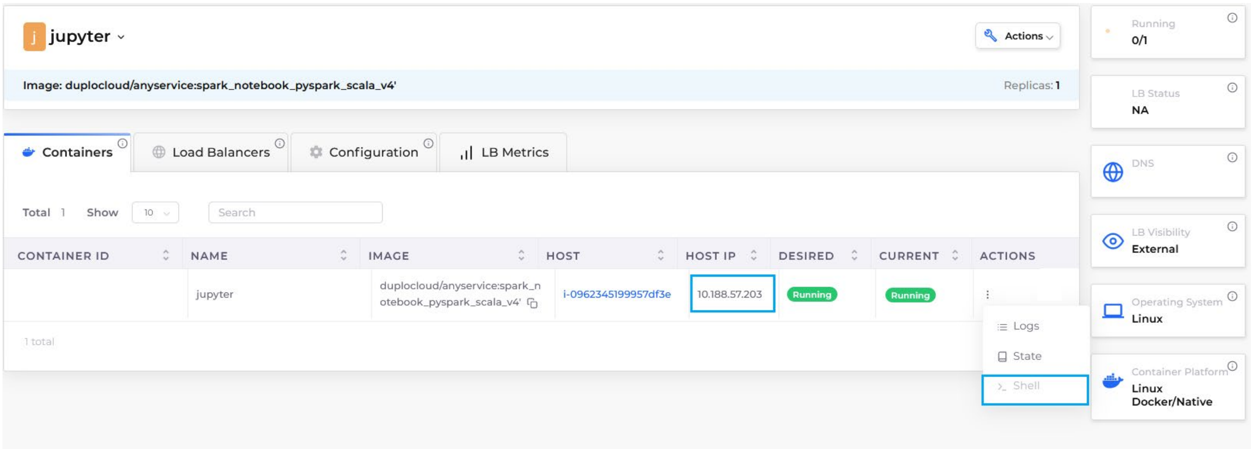
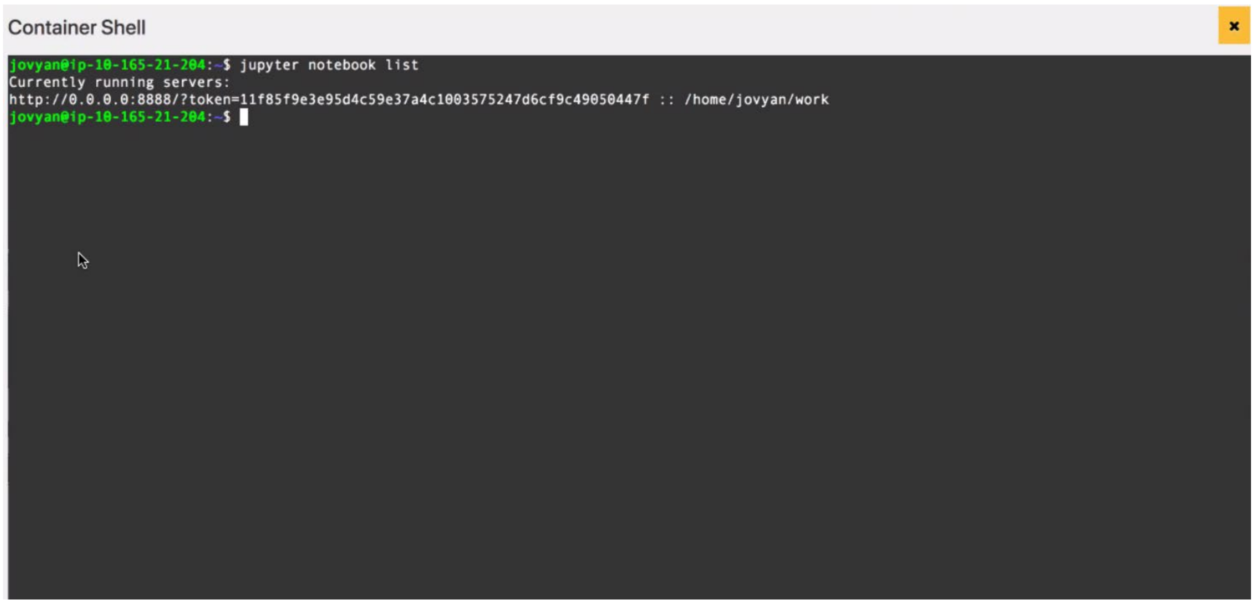


Image: duplocloud/anyservice:spark_notebook_pyspark_scala_v4'   Replicas: 1

CONTAINER ID	NAME	IMAGE	HOST	HOST IP	DESIRED	CURRENT	ACTIONS
	jupyter	duplocloud/anyservice:spark_notebook_pyspark_scala_v4'	i-0962345199957df3e	10.188.57.203	Running	Running	⋮

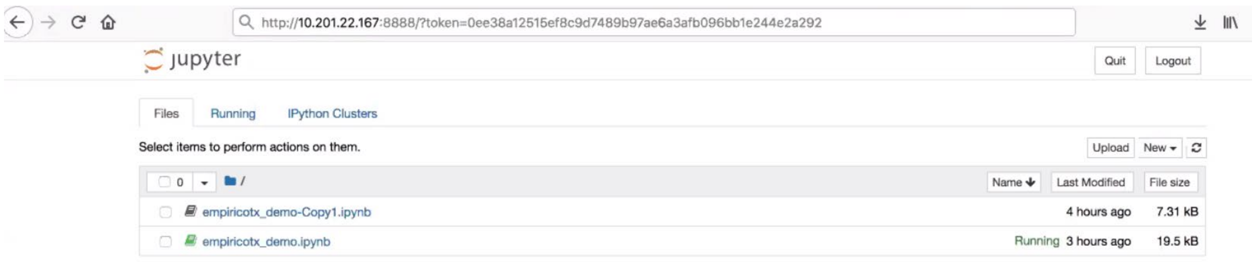
1 total

- **Get Jupyter URL:** Once you are inside the shell. Enter 'jupyter notebook list' to get the URL along with auth token. Replace the Ip with Jupyter ip you copied previously.



```
jovyan@ip-10-165-21-204:~$ jupyter notebook list
Currently running servers:
http://0.0.0.0:8888/?token=11f85f9e3e95d4c59e37a4c1003575247d6cf9c49050447f :: /home/jovyan/work
jovyan@ip-10-165-21-204:~$
```

- **Open Jupyter:** In your browser, invoke the Jupyter URL and you should be able to see the UI.



```
http://10.201.22.167:8888/?token=0ee38a12515ef8c9d7489b97ae6a3afb096bb1e244e2a292
```

Select items to perform actions on them.

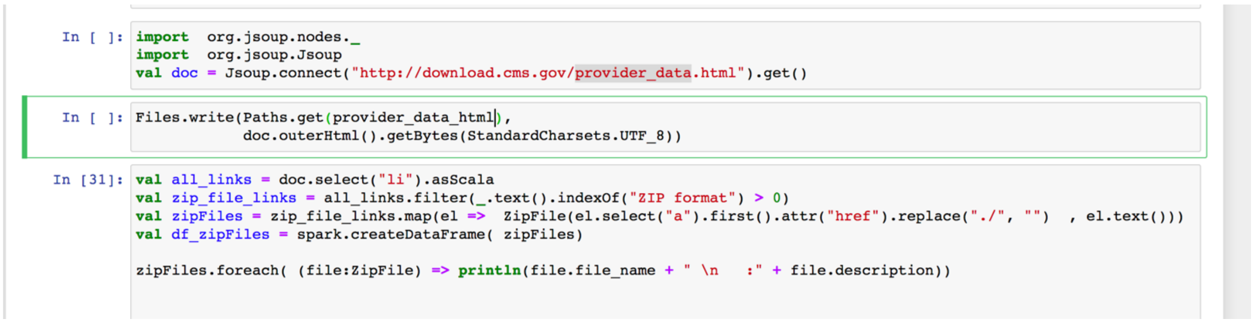
Name		Last Modified	File size
empiricotx_demo-Copy1.ipynb		4 hours ago	7.31 kB
empiricotx_demo.ipynb	Running	3 hours ago	19.5 kB

Now you can use Jupyter to connect to data sources and destinations and do ETL jobs. Sources and destinations can include various SQL & NoSQL databases, S3 and various reporting tools including big data and GPU=based Deep learning.

- **Jupyter with an ETL Use-case**

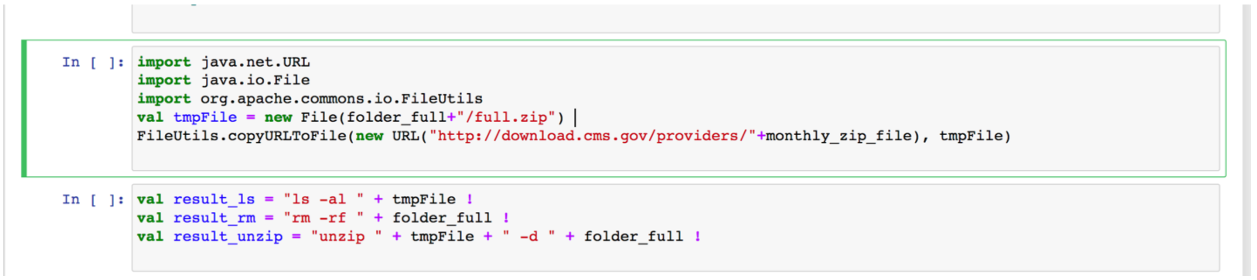
  In this tutorial we will create a Jupyter notebook and show some basic web scraping, using Spark for preprocessing, exporting into schema, do ETLs, join multiple dataframes (parquets), and export reports into MySQL.

  - **Scrape data from Internet:** Connect to a website and parse html (using jsoup)



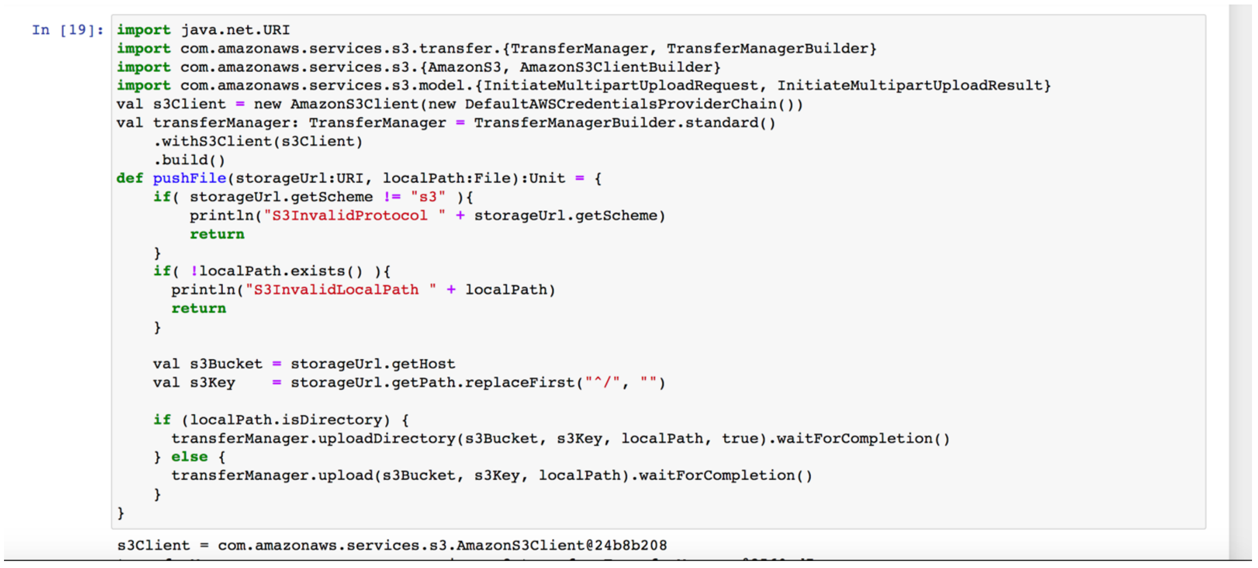
```
In [ ]: import  org.jsoup.nodes._
        import  org.jsoup.Jsoup
        val doc = Jsoup.connect("http://download.cms.gov/provider_data.html").get()

In [ ]: Files.write(Paths.get(provider_data_html),
                    doc.outerHtml().getBytes(StandardCharsets.UTF_8))

In [31]: val all_links = doc.select("li").asScala
         val zip_file_links = all_links.filter(_.text().indexOf("ZIP format") > 0)
         val zipFiles = zip_file_links.map(el =>  ZipFile(el.select("a").first().attr("href").replace("./", "")  , el.text()))
         val df_zipFiles = spark.createDataFrame( zipFiles)

         zipFiles.foreach( (file:ZipFile) => println(file.file_name + " \n    :" + file.description))
```

- **Download data:** Extract the downloaded zip. This particular file is 8 GB in size and has 9 million records in csv



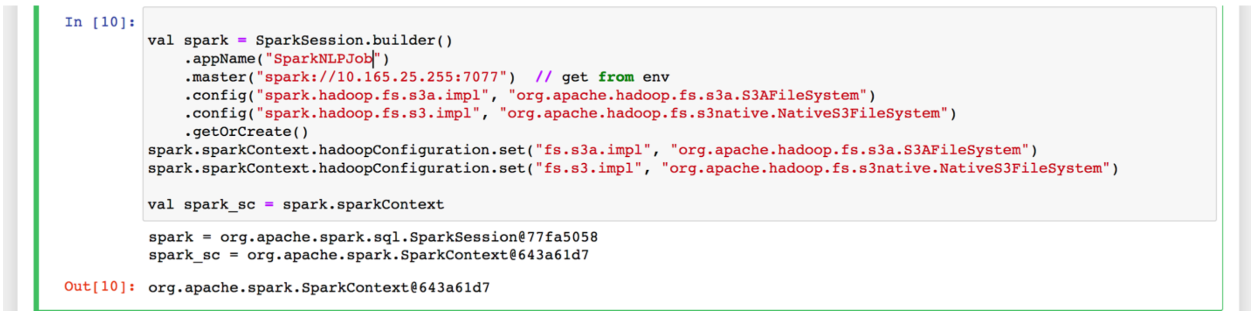
```
In [ ]: import java.net.URL
        import java.io.File
        import org.apache.commons.io.FileUtils
        val tmpFile = new File(folder_full+"/full.zip")
        FileUtils.copyURLToFile(new URL("http://download.cms.gov/providers/"+monthly_zip_file), tmpFile)

In [ ]: val result_ls = "ls -al " + tmpFile !
        val result_rm = "rm -rf " + folder_full !
        val result_unzip = "unzip " + tmpFile + " -d " + folder_full !
```

- **Save to S3** Upload the data to Aws S3



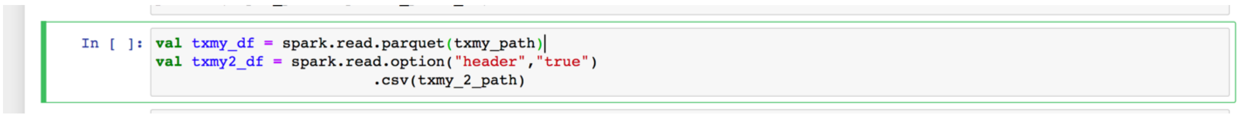
```
In [19]: import java.net.URI
         import com.amazonaws.services.s3.transfer.{TransferManager, TransferManagerBuilder}
         import com.amazonaws.services.s3.{AmazonS3, AmazonS3ClientBuilder}
         import com.amazonaws.services.s3.model.{InitiateMultipartUploadRequest, InitiateMultipartUploadResult}
         val s3Client = new AmazonS3Client(new DefaultAWSCredentialsProviderChain())
         val transferManager: TransferManager = TransferManagerBuilder.standard()
             .withS3Client(s3Client)
             .build()
         def pushFile(storageUrl:URI, localPath:File):Unit = {
             if( storageUrl.getScheme != "s3" ){
                 println("S3InvalidProtocol " + storageUrl.getScheme)
                 return
             }
             if( !localPath.exists() ){
               println("S3InvalidLocalPath " + localPath)
               return
             }

             val s3Bucket = storageUrl.getHost
             val s3Key    = storageUrl.getPath.replaceFirst("^/", "")

             if (localPath.isDirectory) {
               transferManager.uploadDirectory(s3Bucket, s3Key, localPath, true).waitForCompletion()
             } else {
               transferManager.upload(s3Bucket, s3Key, localPath).waitForCompletion()
             }
         }

         s3Client = com.amazonaws.services.s3.AmazonS3Client@24b8b208
```

- **Connect to spark cluster** Also Configure session with required settings to read & write from Aws S3



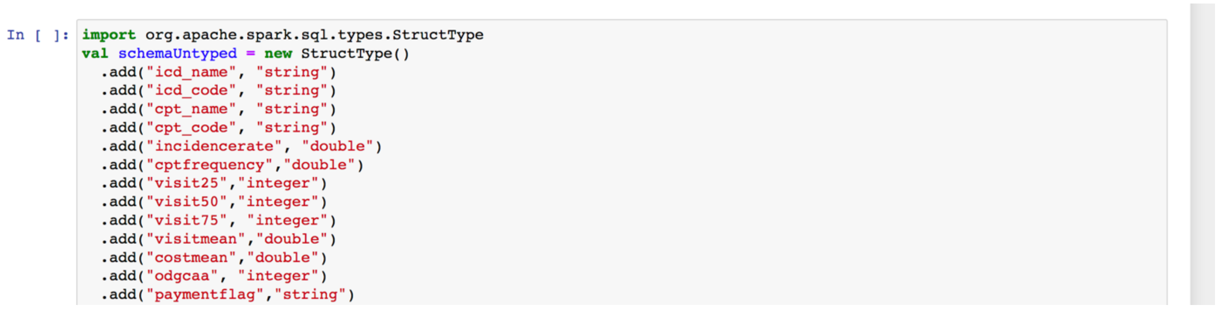
```
In [10]:  val spark = SparkSession.builder()
              .appName("SparkNLPJob")
              .master("spark://10.165.25.255:7077")  // get from env
              .config("spark.hadoop.fs.s3a.impl", "org.apache.hadoop.fs.s3a.S3AFileSystem")
              .config("spark.hadoop.fs.s3.impl", "org.apache.hadoop.fs.s3native.NativeS3FileSystem")
              .getOrCreate()
          spark.sparkContext.hadoopConfiguration.set("fs.s3a.impl", "org.apache.hadoop.fs.s3a.S3AFileSystem")
          spark.sparkContext.hadoopConfiguration.set("fs.s3.impl", "org.apache.hadoop.fs.s3native.NativeS3FileSystem")

          val spark_sc = spark.sparkContext

          spark = org.apache.spark.sql.SparkSession@77fa5058
          spark_sc = org.apache.spark.SparkContext@643a61d7

Out[10]:  org.apache.spark.SparkContext@643a61d7
```

- **Load data in spark cluster**



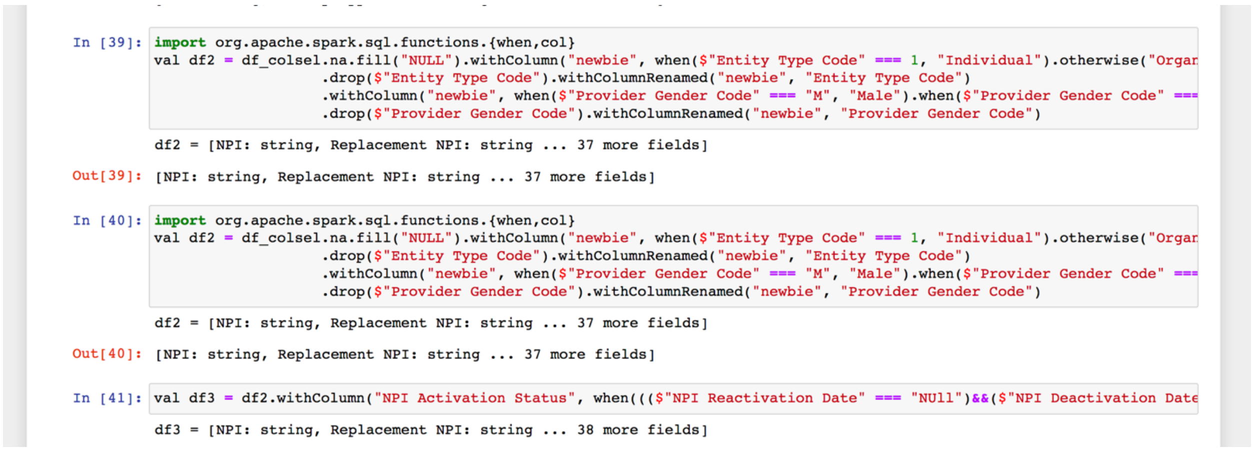
```
In [ ]:  val txmy_df = spark.read.parquet(txmy_path)
         val txmy2_df = spark.read.option("header","true")
                             .csv(txmy_2_path)
```

- **Define schema**



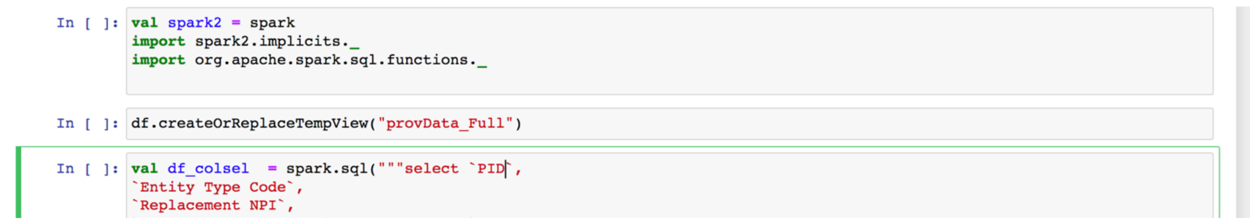
```
In [ ]:  import org.apache.spark.sql.types.StructType
         val schemaUntyped = new StructType()
           .add("icd_name", "string")
           .add("icd_code", "string")
           .add("cpt_name", "string")
           .add("cpt_code", "string")
           .add("incidencerate", "double")
           .add("cptfrequency","double")
           .add("visit25","integer")
           .add("visit50","integer")
           .add("visit75", "integer")
           .add("visitmean","double")
           .add("costmean","double")
           .add("odgcaa", "integer")
           .add("paymentflag","string")
```

- **Do data processing**



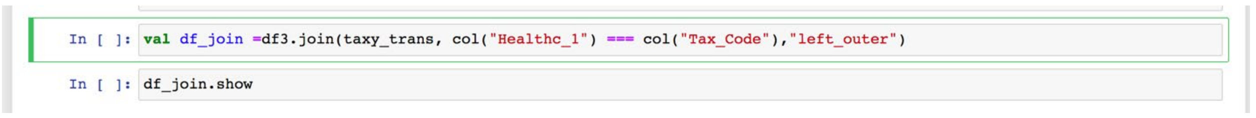
```
In [39]:  import org.apache.spark.sql.functions.{when,col}
          val df2 = df_colsel.na.fill("NULL").withColumn("newbie", when($"Entity Type Code" === 1, "Individual").otherwise("Orgar
                              .drop($"Entity Type Code").withColumnRenamed("newbie", "Entity Type Code")
                              .withColumn("newbie", when($"Provider Gender Code" === "M", "Male").when($"Provider Gender Code" ===
                              .drop($"Provider Gender Code").withColumnRenamed("newbie", "Provider Gender Code")

          df2 = [NPI: string, Replacement NPI: string ... 37 more fields]

Out[39]:  [NPI: string, Replacement NPI: string ... 37 more fields]

In [40]:  import org.apache.spark.sql.functions.{when,col}
          val df2 = df_colsel.na.fill("NULL").withColumn("newbie", when($"Entity Type Code" === 1, "Individual").otherwise("Orgar
                              .drop($"Entity Type Code").withColumnRenamed("newbie", "Entity Type Code")
                              .withColumn("newbie", when($"Provider Gender Code" === "M", "Male").when($"Provider Gender Code" ===
                              .drop($"Provider Gender Code").withColumnRenamed("newbie", "Provider Gender Code")

          df2 = [NPI: string, Replacement NPI: string ... 37 more fields]

Out[40]:  [NPI: string, Replacement NPI: string ... 37 more fields]

In [41]:  val df3 = df2.withColumn("NPI Activation Status", when((($"NPI Reactivation Date" === "NUll")&&($"NPI Deactivation Date

          df3 = [NPI: string, Replacement NPI: string ... 38 more fields]
```

- **Spark SQL**



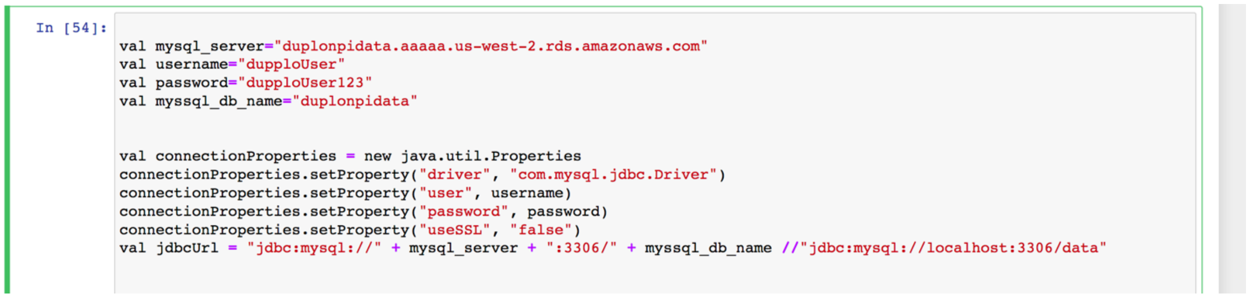
```
In [ ]: val spark2 = spark
        import spark2.implicits._
        import org.apache.spark.sql.functions._

In [ ]: df.createOrReplaceTempView("provData_Full")

In [ ]: val df_colsel  = spark.sql("""select `PID`,
        `Entity Type Code`,
        `Replacement NPI`,
```

- **Spark SQL joins** 20 GB of data from multiple sources



```
In [ ]: val df_join =df3.join(taxy_trans, col("Healthc_1") === col("Tax_Code"),"left_outer")

In [ ]: df_join.show
```

- **Export reports to RDS for UI consumption** Generate various charts and graphs



```
In [54]:
val mysql_server="duplonpidata.aaaaa.us-west-2.rds.amazonaws.com"
val username="dupploUser"
val password="dupploUser123"
val myssql_db_name="duplonpidata"


val connectionProperties = new java.util.Properties
connectionProperties.setProperty("driver", "com.mysql.jdbc.Driver")
connectionProperties.setProperty("user", username)
connectionProperties.setProperty("password", password)
connectionProperties.setProperty("useSSL", "false")
val jdbcUrl = "jdbc:mysql://" + mysql_server + ":3306/" + myssql_db_name //"jdbc:mysql://localhost:3306/data"
```

Security & Compliance

Refer to Compliance

Cloud Migration (SMS)

Documentation TBD. Please contact DuploCloud team for assistance.

Remote Working

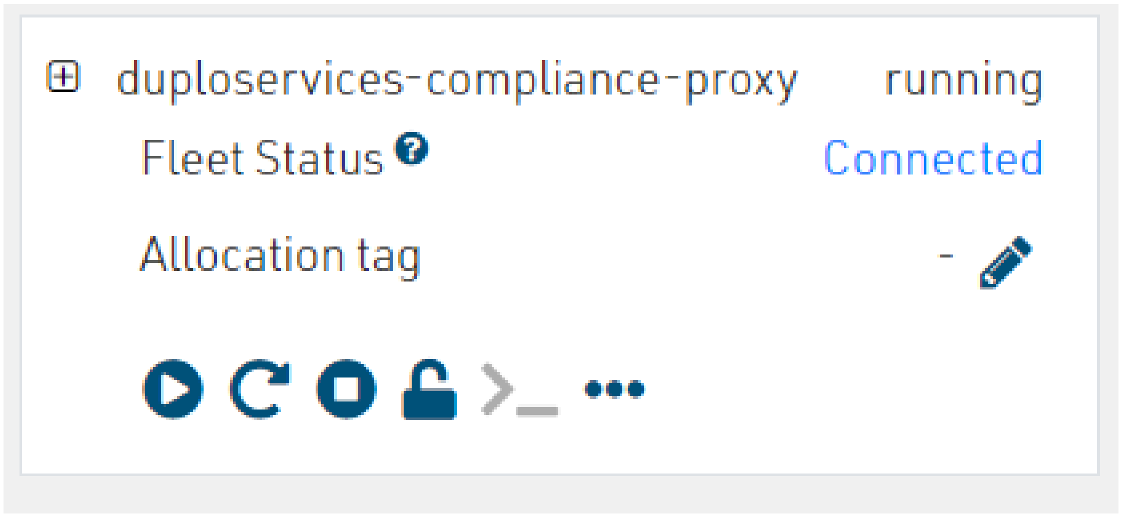
Refer to Workspaces

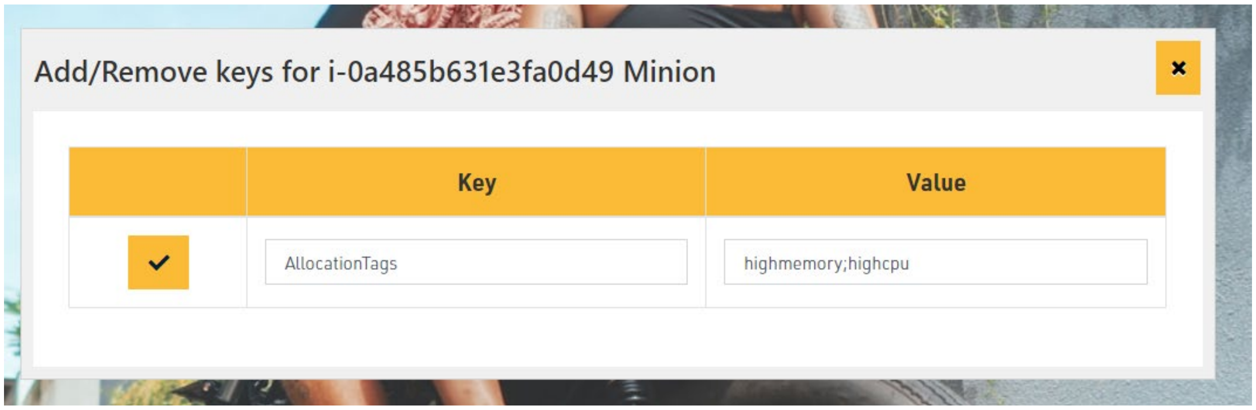
Additional Deployment functionalities

Advanced Functions

- **Allocation tags:** By default, DuploCloud spreads the container replicas across the hosts. By virtue of allocation tag the user can pin a container to a set of hosts that have a specific tag. Click on the edit icon next to allocation tag under host.

"`AllocationTags`" as key is already pre-populated, configure the value of your choice for example "`highmemory;highcpu`" or "`serviceA`" and click on save on the left. Then, when creating a service, set the allocation tag value to be a substring of the above tag, for example `highmemory` or `serviceA`. The allocation algorithm tries to pick a host where `AllocationTag` specified in the service is a substring of the `AllocationTags` of the host.



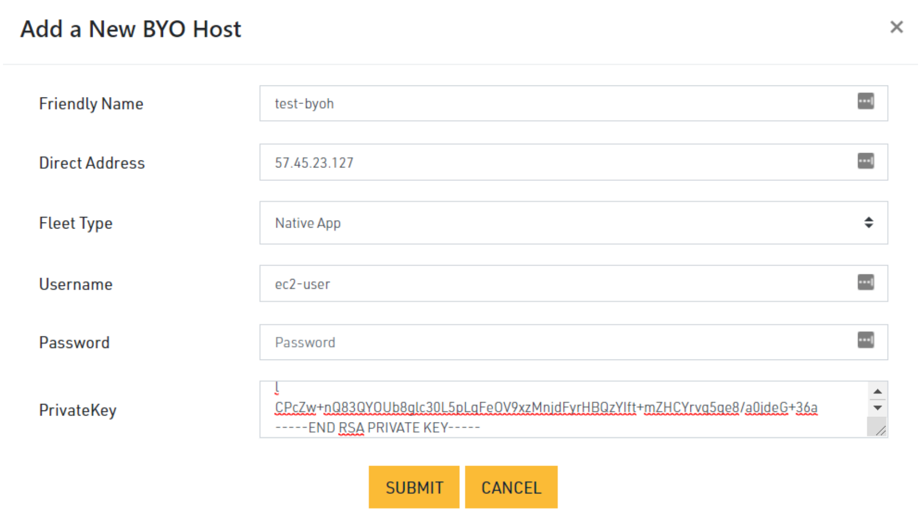

Add/Remove keys for i-0a485b631e3fa0d49 Minion

	Key	Value
✓	AllocationTags	highmemory;highcpu

BYOH

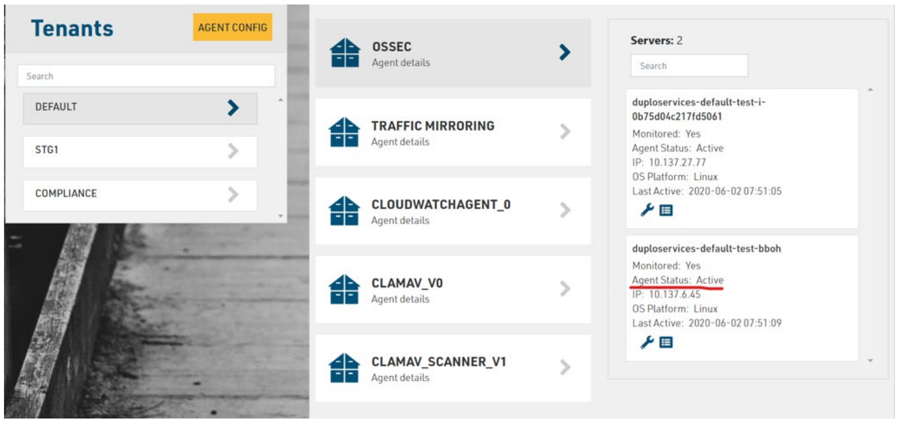
If you have a host already running somewhere in the cloud or on-premise, you can bring that to DuploCloud using BYOH functionality and let DuploCloud manage the host, let it be running the containers or installing the compliance agents. To configure BYOH go to **Deployments → Hosts → BYO-HOSTS**. Click on **add** and provide the name, IP Address and in the Fleet type select

Native App (if you don't, DuploCloud will not manage the containers but manage the compliance agents on the host), provide the username, password/Private key file. Make sure the SSH access to the host is opened to access from the DuploCloud. After the host is added in about 5mins you can go to **Security → Agents → Select Agent** and see that the agent on host is in Active state.



Add a New BYO Host

Friendly Name	test-byoh
Direct Address	57.45.23.127
Fleet Type	Native App
Username	ec2-user
Password	Password
PrivateKey	l CPcZw+nQ83QYOUb8glc30L5pLqFeOV9xzMnjdFyrHBQzYIft+mZHCYrvq5qe8/a0jdeG+36a -----END RSA PRIVATE KEY-----



Servers: 2

duploservices-default-test-i-0b75d04c217fd5061
Monitored: Yes
Agent Status: Active
IP: 10.137.27.77
OS Platform: Linux
Last Active: 2020-06-02 07:51:05

duploservices-default-test-bboh
Monitored: Yes
Agent Status: Active
IP: 10.137.6.45
OS Platform: Linux
Last Active: 2020-06-02 07:51:09

Administrator's Guide

Configuration Scopes

- **Base Configuration:** This must be set up by the user outside DuploCloud and provided as an input to DuploCloud. This is a once in a lifetime configuration. This configuration includes one VPC and one subnet with appropriate routes. It is recommended that you setup at least 2 Availability Zones, one public and one private subnet in each availability zone. All resources (EC2 instances, RDS, Elastic-cache, etc.) are placed in the private subnet by default. ELB is placed in public or private subnet based on user (tenant) choice. DuploCloud provides a default cloud formation template that can be used if desired.

- **Plan Configuration:** Every tenant is part of one and only one plan. Configuration applied at the plan level is applies to all tenants in a plan. A plan can be used to denote say a dev environment, a class of tenants (private\public facing) etc. Following are the plan configurations. Except of VPC and subnets rest of the parameters can be changed at will. Plan parameters include:

```
{
                   "Name": "devplan", /* Name of the plan */
                   "Images": [ /* AMIs that are made available for the
tenants */
                    {
                     "Name": "Duplo-Host-Docker-v1.17", /* User
friendly Name of the AMI displayed to the user */
                     "ImageId": "ami-0861ea68", /* AWS AMI ID */
                     "OS": "Linux"
                    },
                    {
                     "Name": "ubuntu-dev",
                     "ImageId": "ami-d83af7b8",
                     "OS": "Linux"
                    },
                    {
                     "Name": "Windows-Docker-Fleet",
                     "ImageId": "ami-1977d979",
                     "OS": "Windows"
                    },
                    {
                     "Name": "Unmanaged Ubuntu-16.04",
                     "ImageId": "ami-5e63d13e",
                     "OS": "Linux"
                    }
                   ],
                   "Quotas": [ /* Limit of on the number of resources
that be used by each tenant. If none is set for a given resource type,
then there is no limit. */
                    {
                     "ResourceType": "ec2",
                     "CumulativeCount": 2,
                     "InstanceQuotas": [
                      {
                       "InstanceType": "t2.medium",
                       "MetaData": "(2CPU, 4GB)",
                       "Count": 2
                      },
                      {
                       "InstanceType": "t2.small",
                       "MetaData": "(1CPU, 2GB)",
                       "Count": 2
                      }
```

```
  ]
 },
 {
  "ResourceType": "rds",
  "CumulativeCount": 1,
  "InstanceQuotas": [
   {
    "InstanceType": "db.t2.small",
    "MetaData": "(1CPU, 1.7GB)",
    "Count": 1
   }
  ]
 },
 {
  "ResourceType": "ecache",
  "CumulativeCount": 1,
  "InstanceQuotas": [
   {
    "InstanceType": "cache.t2.small",
    "MetaData": "(1CPU, 1.7GB)",
    "Count": 1
   }
  ]
 },
 {
  "ResourceType": "s3",
  "CumulativeCount": 1,
  "InstanceQuotas": [
   {
    "InstanceType": "bucket",
    "MetaData": "bucket",
    "Count": 1
   }
  ]
 },
 {
  "ResourceType": "sqs",
  "CumulativeCount": 1,
  "InstanceQuotas": [
   {
    "InstanceType": "queue",
    "MetaData": "queue",
    "Count": 1
   }
  ]
 },
 {
  "ResourceType": "dynamodb",
  "CumulativeCount": 1,
  "InstanceQuotas": [
   {
    "InstanceType": "table",
    "MetaData": "table",
    "Count": 1
   }
  ]
 },
                        {
                         "ResourceType": "elb",
                         "CumulativeCount": 2,
                         "InstanceQuotas": [
                           {
                            "InstanceType": "elb",
                            "MetaData": "elb",
                            "Count": 1
                           }
                          ]
                        },
                        {
                         "ResourceType": "sns",
                         "CumulativeCount": 1,
                         "InstanceQuotas": [
                           {
                            "InstanceType": "topic",
                            "MetaData": "topic",
                            "Count": 1
                           }
                          ]
                        }
                       ],
                       "AwsConfig": {
                        "VpcId": "vpc-1eafce79", /* VPC for this tenant */
                        "AwsHostSg": "sg-c2d899ba", /* list of security
groups separated by ;. All hosts will be placed in this security
groups. */
                        "AwsElbSg": "sg-b9d899c1", /* Security group in
which the ELB will be placed. This applies to traffic into the extenal
(VIP) of the elb. Internal Sg between hosts and ELB will be auo setup.
*/
                        "AwsPublicSubnet": "subnet-0066b449;subnet-
24e25943", /* Subnets in which hpublic facing ELBs must be placed. Each
subnet corresponds to an AZ. */
                        "AwsInternalElbSubnet": "subnet-0e66b447;subnet-
23e25944", /* Subnets in which internal resources like Ec2 hosts, rds
ecaache etc will be placed. If you like hosts to be public facing then
put the same public subnets here. */
                        "AwsElastiCacheSubnetGrp": "duplo-cache-
aww6gk5hsy4n",
                        "AwsRdsSubnetGrp": "duplo-vpc-21-resources-
dbsubnetduplo-z9fh3g59362z",
                        "CommonPolicyARNs": "", /*Set of IAM policy ARNs
that will be applied to all tenants. */
                        "Domain name": "" /* Name of route 53  domain that
will be used for tenant custom dns names*/
                        "CertMgrResourceArns": "" /* List of certificate
arns that wil be available to the tenant for SSL termination on the
ELB. */
                       },
                       "UnrestrictedExtLB": false,
                       "Capabilities": {
                        "DisableNativeApps": false,
                        "DisablePublicEps": false,
                        "DisablePrivateElb": false,
                        "AssignInstanceElasticIp": false,
                    "BlockEbsOptimization": false,
                    "DisableSumoIntegration": false,
                    "DisableSignalFxIntegration": false,
                    "EnableTenantExpiry": false
                  }
                }
```

- **Tenant Configuration:** These are the configuration that we covered in the deployment guide.

Access Control

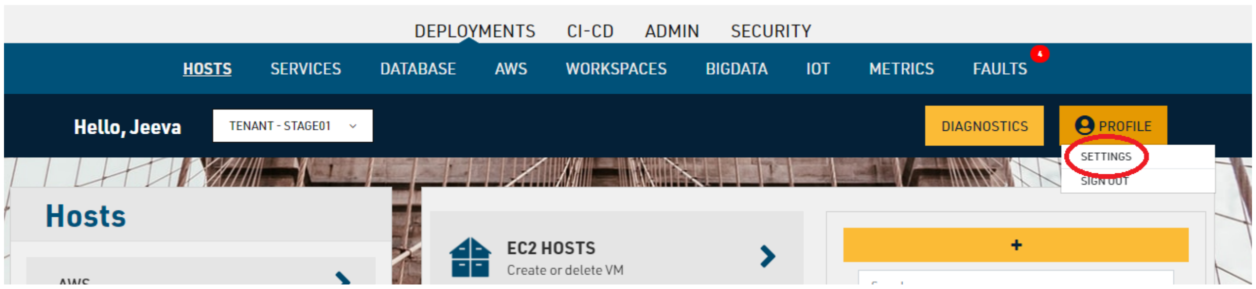
There are 2 types of roles, user and administrator. Each user can have access to one or more tenants. Each tenant can be accessed by one or more users. Administrator has access to all tenants plus the administrative functions like plan configuration, system dashboard, system faults, etc.

Accessing Cloud Server

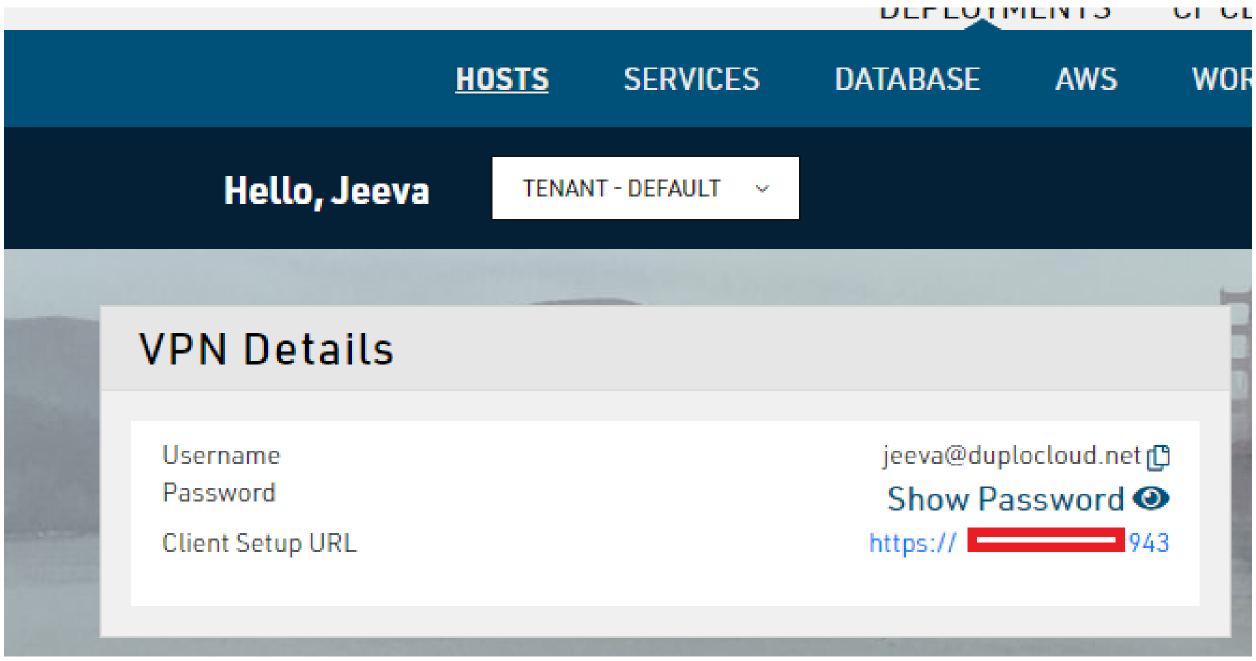
VPN Access

DuploCloud provisions the OpenVPN server for users to connect with the cloud resources like EC2, RDS instances, Elastic Search, Elastic Cache etc. Below steps will guide you to connect with VPN server using OpenVPN client.

- **Setup OpenVPN client**
  Login to OpenVPN web portal to download OpenVPN config and client.
- Click on **Profile → Settings** as shown in image below



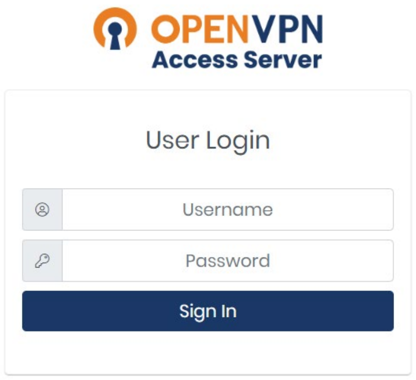
- Click the openvpn link which will open a new tab.



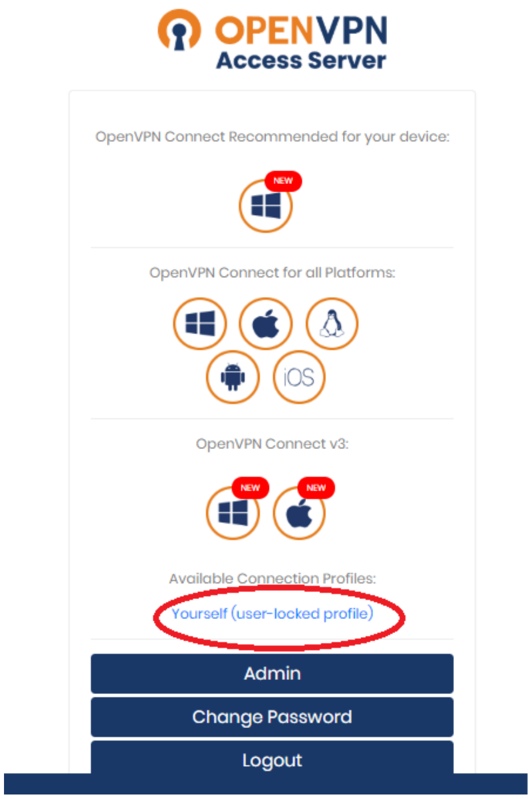
VPN Details

Username	jeeva@duplocloud.net
Password	Show Password
Client Setup URL	https:// [redacted] 943

- Login using Username and Password provided.



- Download the openvpn config and client.

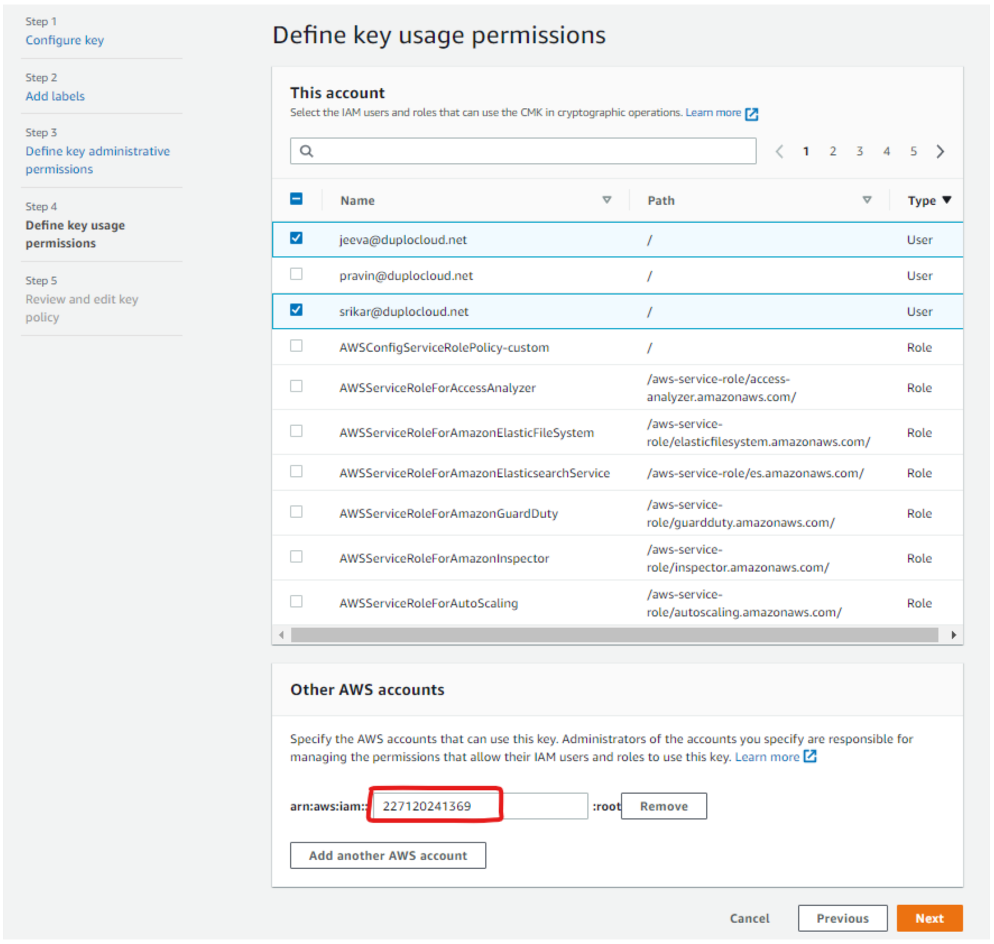

OPENVPN
Access Server
OpenVPN Connect Recommended for your device:
NEW
OpenVPN Connect for all Platforms:
iOS
OpenVPN Connect v3:
NEW
NEW
Available Connection Profiles:
Yourself (user-locked profile)
Admin
Change Password
Logout

- Now you are ready to connect with VPN using OpenVPN client.

Sharing Encrypted DB

Sharing Unencrypted DB to other accounts is very simple and straight forward. But sharing encrypted DB is slightly difficult. Here we will go through the steps that needs to be followed to share the encrypted DB:

- Create a new customer managed key in AWS KMS, in the Define key usage permissions provide the account id of the other account.



Step 1
Configure key
Step 2
Add labels
Step 3
Define key administrative permissions
Step 4
Define key usage permissions
Step 5
Review and edit key policy

Define key usage permissions

This account

Select the IAM users and roles that can use the CMK in cryptographic operations. Learn more

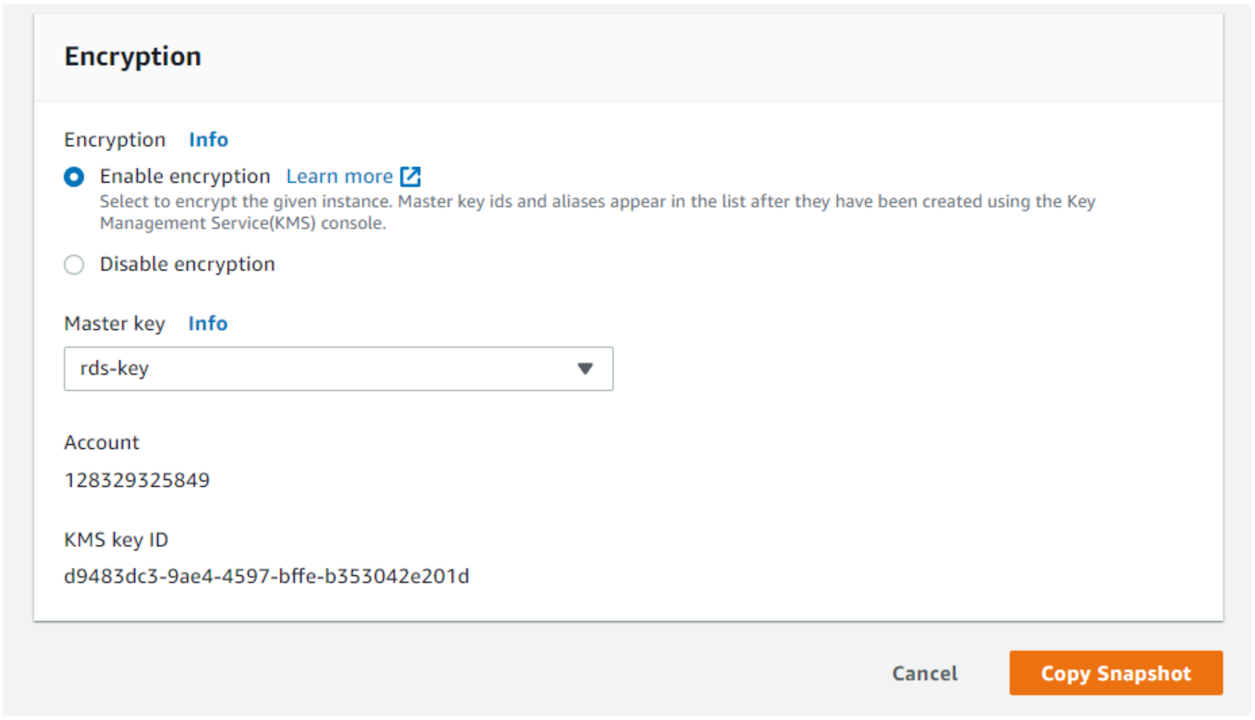
	Name	Path	Type
✓	jeeva@duplocloud.net	/	User
	pravin@duplocloud.net	/	User
✓	srikar@duplocloud.net	/	User
	AWSConfigServiceRolePolicy-custom	/	Role
	AWSServiceRoleForAccessAnalyzer	/aws-service-role/access-analyzer.amazonaws.com/	Role
	AWSServiceRoleForAmazonElasticFileSystem	/aws-service-role/elasticfilesystem.amazonaws.com/	Role
	AWSServiceRoleForAmazonElasticsearchService	/aws-service-role/es.amazonaws.com/	Role
	AWSServiceRoleForAmazonGuardDuty	/aws-service-role/guardduty.amazonaws.com/	Role
	AWSServiceRoleForAmazonInspector	/aws-service-role/inspector.amazonaws.com/	Role
	AWSServiceRoleForAutoScaling	/aws-service-role/autoscaling.amazonaws.com/	Role

Other AWS accounts

Specify the AWS accounts that can use this key. Administrators of the accounts you specify are responsible for managing the permissions that allow their IAM users and roles to use this key. Learn more

arn:aws:iam:: 227120241369 :root

- Once the key is created, go to **RDS → Snapshots**, select the snapshot and click Copy Snapshot. In the encryption change the master key to the key we created before.


Encryption

Encryption Info

[✓ Enable encryption] Learn more
Select to encrypt the given instance. Master key ids and aliases appear in the list after they have been created using the Key Management Service(KMS) console.

Disable encryption

Master key Info

rds-key

Account

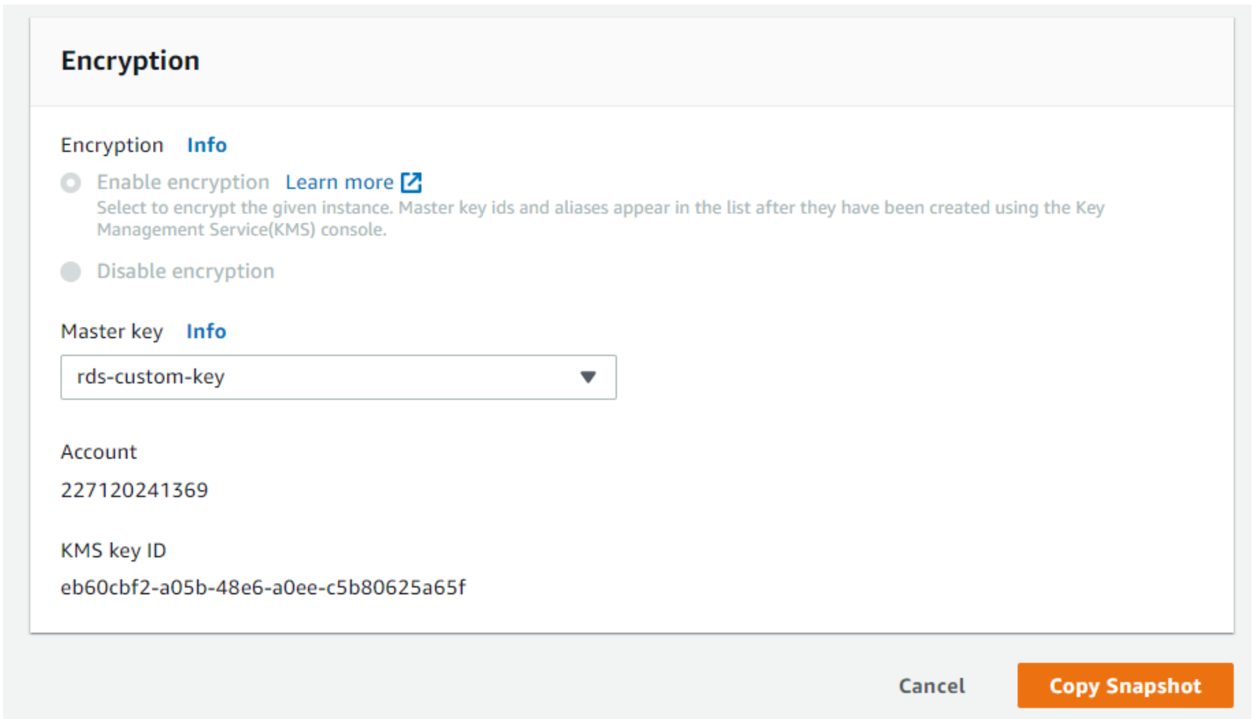
128329325849

KMS key ID

d9483dc3-9ae4-4597-bffe-b353042e201d

- Once the copied snapshot is ready, as usual share the snapshot to another account by clicking share snapshot and providing the other account id.

- 4. Now go to the other **AWS account → RDS → Shared with me**. Select the shared snapshot and click copy-snapshot again and change the encryption key to the encryption key in the account.


Encryption

Encryption Info

Enable encryption Learn more
Select to encrypt the given instance. Master key ids and aliases appear in the list after they have been created using the Key Management Service(KMS) console.

Disable encryption

Master key Info

rds-custom-key

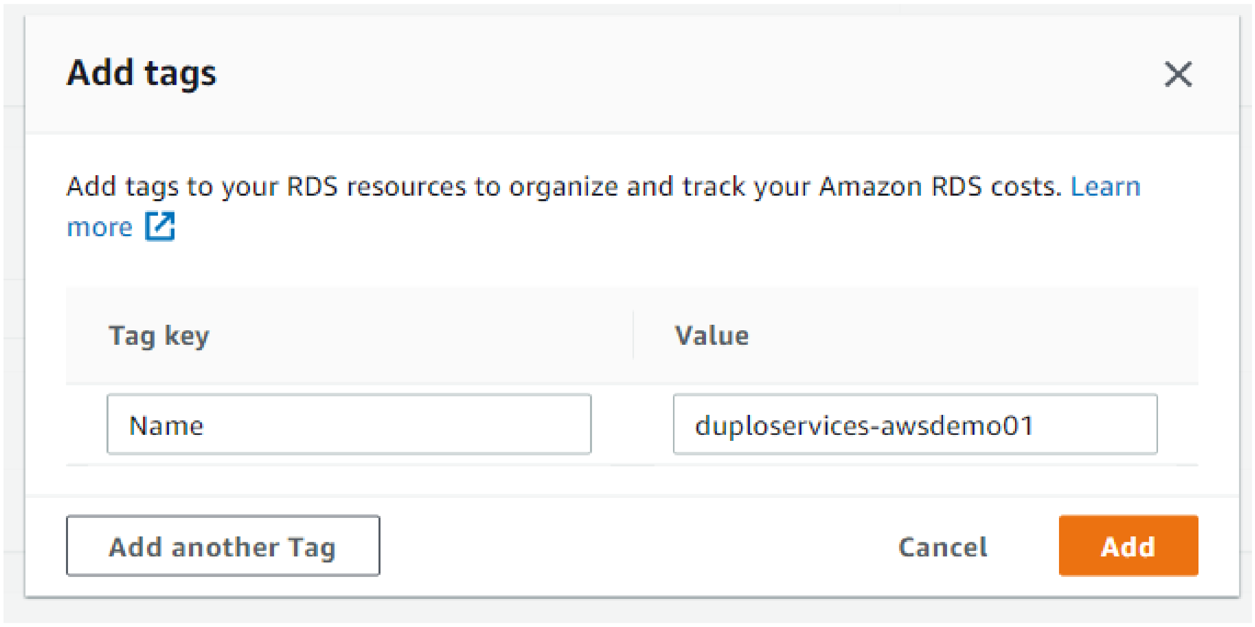
Account

227120241369

KMS key ID

eb60cbf2-a05b-48e6-a0ee-c5b80625a65f

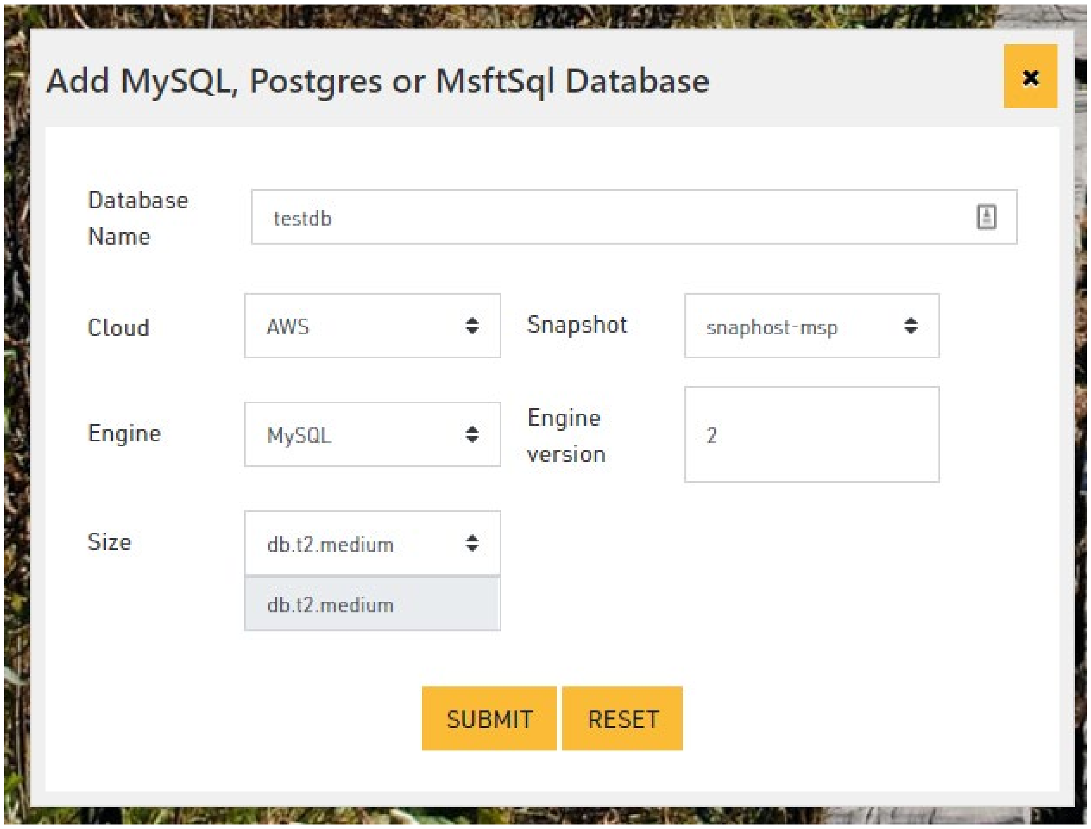
- In the copied snapshot add a tag with Key as "Name" and Value as "duploservices-{tenantname}" where tenantname is the tenant where u want to launch an RDS with this snapshot.


Add tags

Add tags to your RDS resources to organize and track your Amazon RDS costs. Learn more

Tag key	Value
Name	duploservices-awsdemo01

- Go to DuploCloud portal select the tenant. **Open RDS → Add new DB (+ icon) → Give** name for the new DB. In the snapshot select the new snapshot. Enter instance type and hit submit. In few mins, the DB will be created with the data from the snapshot. You must use the existing username and password to access the DB


Add MySQL, Postgres or MsftSql Database

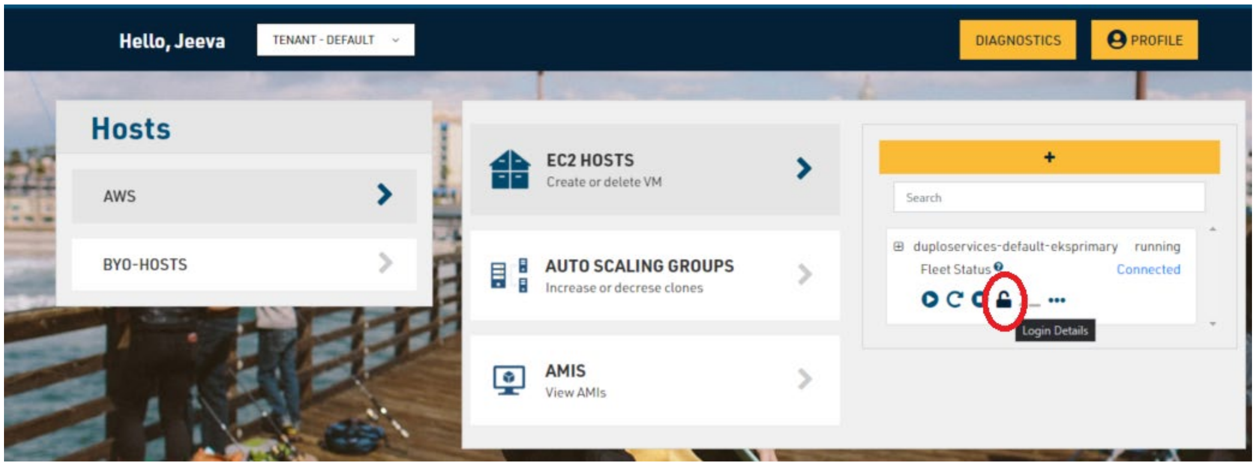
Database Name	testdb		
Cloud	AWS	Snapshot	snaphost-msp
Engine	MySQL	Engine version	2
Size	db.t2.medium		

SSH EC2 Instance

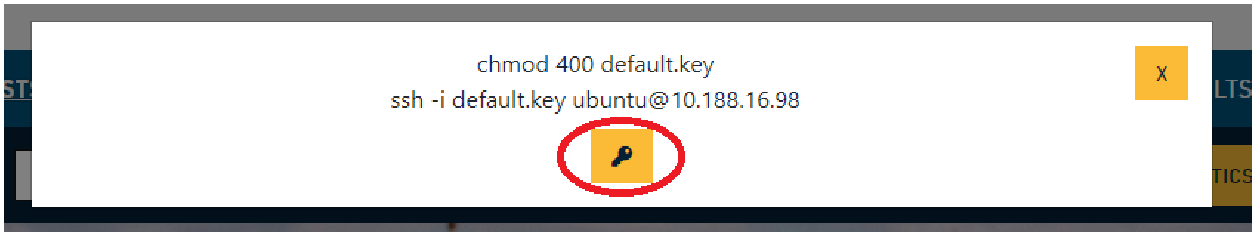
Once you are connected to VPN. You can use the privatekey to SSH into EC2 instance.

- **Download Private key**
  Select Tenant and navigate to Deployments --> Hosts.

- Click on the unlock icon as shown below.



- Click on the key icon to download the private key. Change the permission to key file as shown and ssh into the ec2 instance.



```
chmod 400 default.key
ssh -i default.key ubuntu@10.188.16.98
```

- Now you should be connected to the server.

CI/CD Guide – Katkit: DuploCloud's CI/CD Component

DuploCloud provides a CI/CD framework that allows you to build, test and deploy your application from Git HUB commits and PRs. We call it Katkit. Katkit is a arbitrary code execution engine which allows the user to run arbitrary code before and after deployment. Katkit follows the same notion of a "Tenant" or environment. Thus, tying together CI and CD. In other words, the tests are run against the application in same underlying AWS topology where one's code is running as against running them in a separate fleet of servers which does not capture the interactions of the application with the AWS infrastructure like IAM, Security groups ELB etc.

At a high level, Katkit functions as follows:

- A repository is linked to a Tenant.
- User chooses a GIT commit to run test and deploy

- Katkit deploys a service in the same tenant with the docker image provided by DuploCloud, which is essentially like a jenkins worker and had the Katkit agent in it.
- Katkit agent in the CI container checks out the code at that commit inside the container. It then executes ci.sh from the checked-out code. Essentially each build is a short-lived service that is removed once the ci.sh execution is over.
- User can put any arbitrary code in ci.sh
- Katkit allows, for a given run of a commit, the user to execute code in "phases" where in each phase Katkit repeats the above steps with a difference in the ENV variables that are set in each phase. The code inside ci.sh is to read the env variables and perform different actions corresponding to each phase
- Katkit has a special phase called "deployment" where it does not run ci.sh but it looks for the servicedescription.js file (details below), replaces the docker image tag and replaces it with the git commit sha. It is assumed that the user, before invoking the deployment phase, has gone through a prior phase where he build a docker image which was tagged with the git commit sha. The sha is available as an ENV variable in every phase.

First Deployment

Before using CI/CD, the first deployment of the application needs to be done via DuploCloud menus described above. Make sure that the application works as expected. Katkit is used only for upgrades of container images and run tests that can be written to run before and after.

Environments

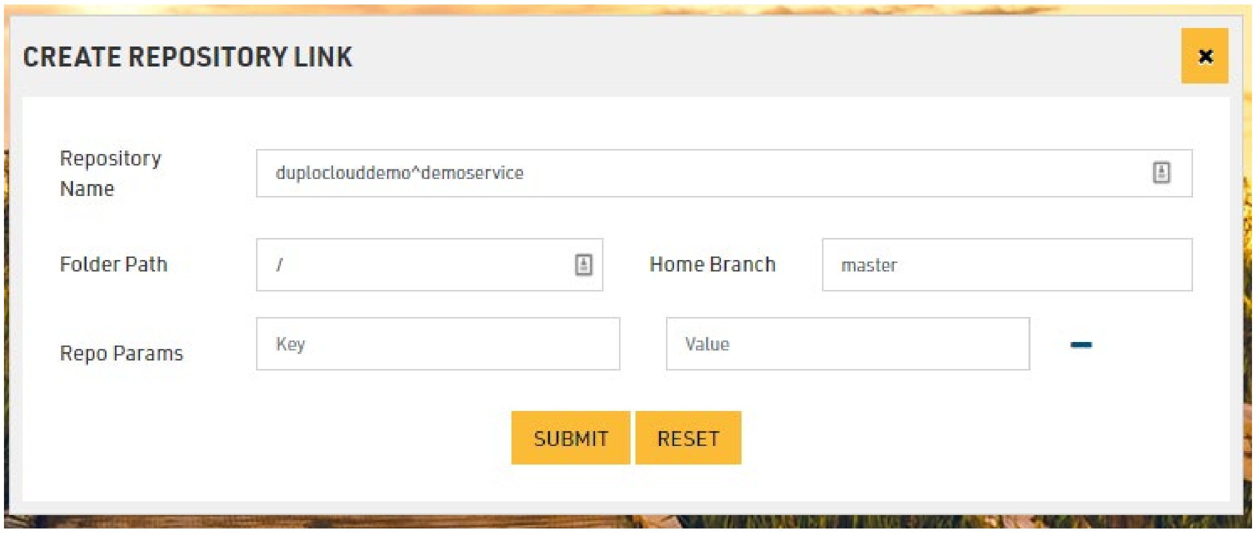
In DuploCloud a standard practice is to have a separate tenant for a given logical application for each deployment environment. For example, say an application called taskrunner would be created as three tenants called d-taskrunner, b-taskrunner and p-taskrunner to represent dev, beta and prod environment. In each tenant one can specify an arbitrary name for the env say "DEV" in the menu Dashboard-->ENV. This string will be set by Katkit as an ENV variable when it runs the tests during CI/CD and thus your test code can use this value to determine what tests should be run in each env or for that matter take any other action.

Export Service Description

Service Description represents the topology of a service. It is a JSON file that is used by Katkit to upgrade the running service. Go to **Deployment → Services → Export**. This will give a json file. Save this as servicedescription.js under the servicedescription folder that must exist at the root of your repository. In this file search for "DockerImage": and here change the image tag to the word <hubtag> for example change "DockerImage": "nginx:latest" to "DockerImage": "nginx:<hubtag>". Remove the ExtraConfig and Replicas field from the file. These have env variables and replicas which would vary from one environment to other. Hence during deployment Katkit will retain what is already present in the current running service.

Link Repository

Once the above steps have been performed, you can link your GitHub Repository to your tenant. In addition to the repository name, you also need to specify the "Home Branch" which is the branch for which the PRs will be monitored by Katkit for the user to run deployments. Same repository and branch combination can be linked in several tenants. If your repository has several services for different tenants, then each service can be represented by a separate folder at the root. This is Folder Path field. Katkit looks for service description file under /servicedescription/servicedescription.js Same repository but different folders can also be used in different tenant. Same tenant can also have different repositories.

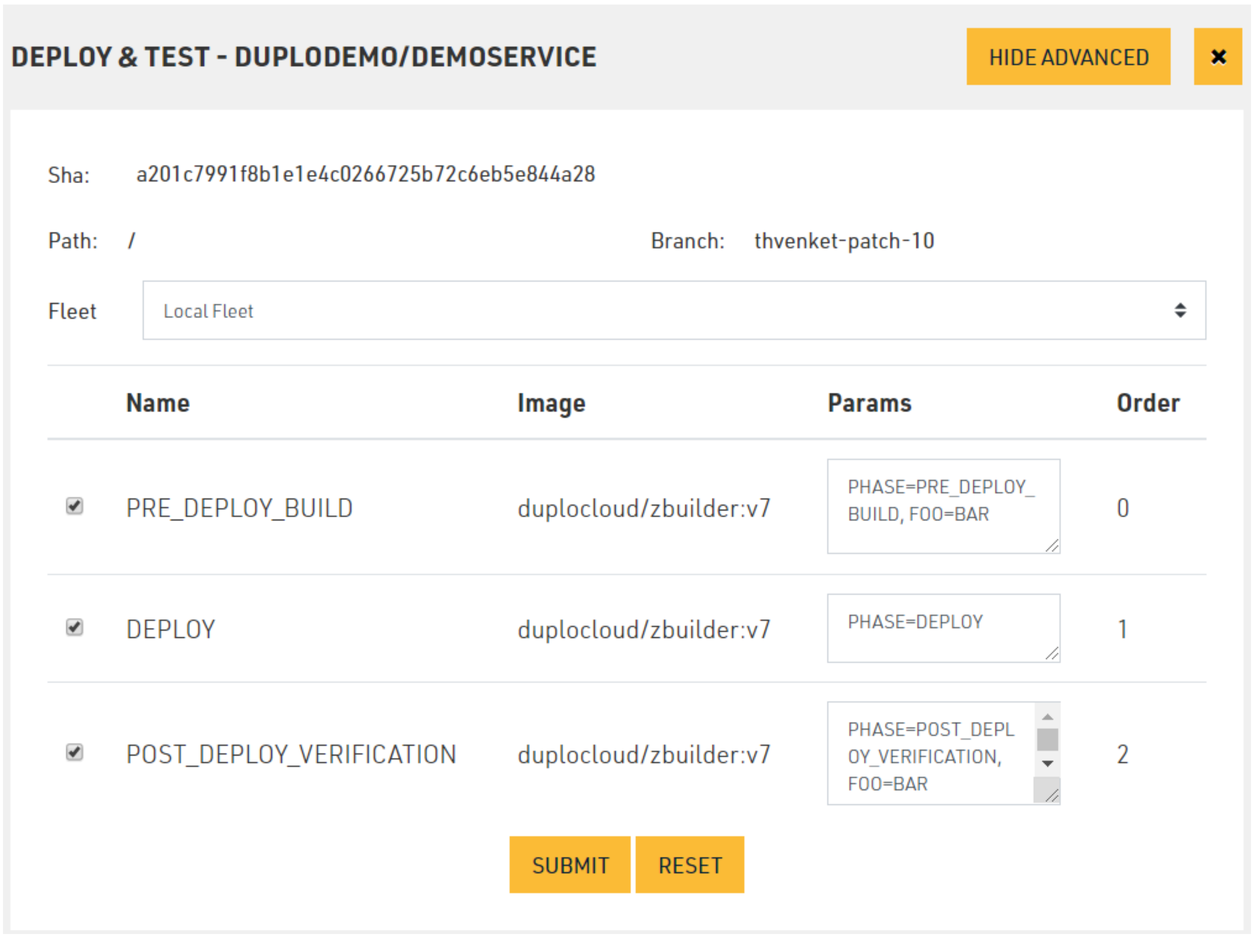


CREATE REPOSITORY LINK

Repository Name	duploclouddemo^demoservice
Folder Path	/
Home Branch	master
Repo Params	Key / Value

Phases

Each CI/CD run comprises of one or more phases. There are two types of phases - execution and deployment. In execution phase Katkit will invoke ci.sh file from the repository. The difference between two execution phases is in ENV variables based on which user code in ci.sh can operate differently. There can be only one deployment phase in each run. Katkit does not run ci.sh in deployment phase but it looks for the servicdescription.js file (details below), replaces the docker image tag <hubtag> and replaces it with the git commit sha. It is assumed that the user, before invoking the deployment phase, has gone through a prior phase where they build a docker image which was tagged with the git commit sha. The sha is available as an ENV variable in every phase.



Sha: a201c7991f8b1e1e4c0266725b72c6eb5e844a28

Path: /

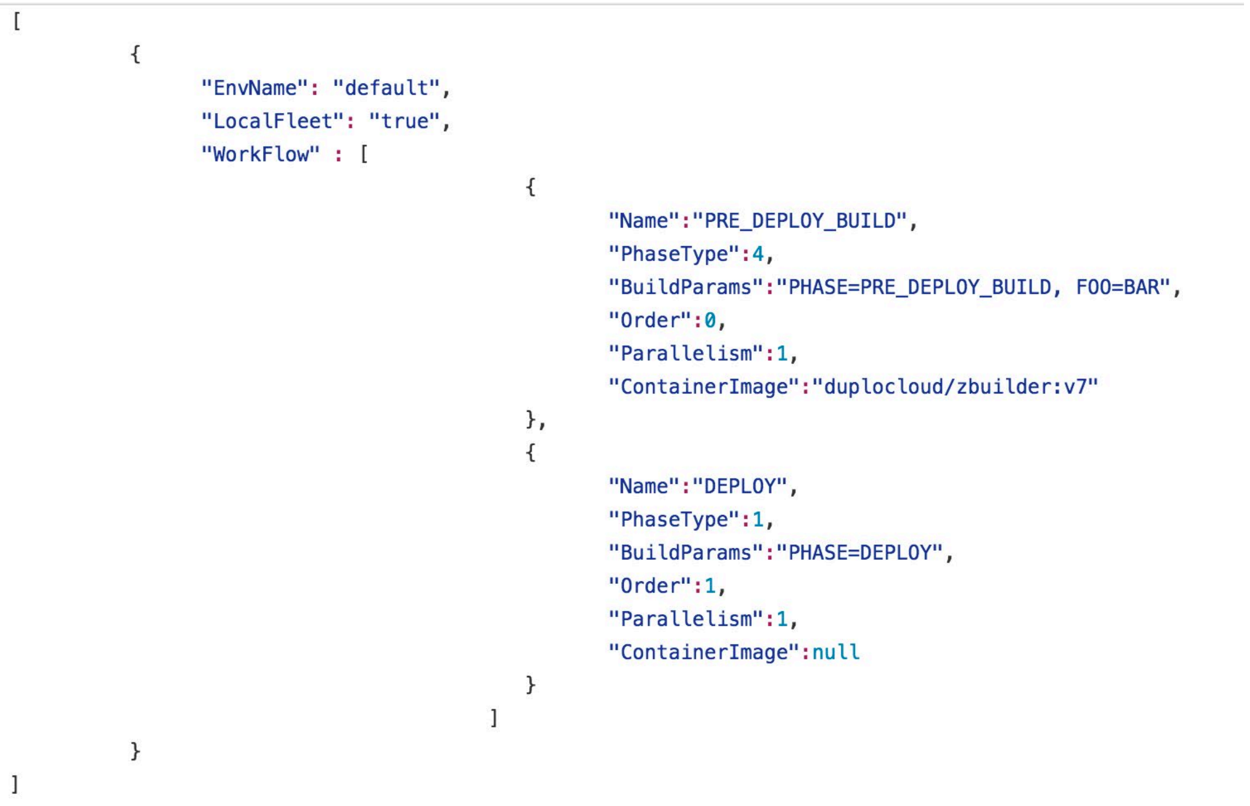
Branch: thvenket-patch-10

Fleet: Local Fleet

	Name	Image	Params	Order
✓	PRE_DEPLOY_BUILD	duplocloud/zbuilder:v7	PHASE=PRE_DEPLOY_BUILD, FOO=BAR	0
✓	DEPLOY	duplocloud/zbuilder:v7	PHASE=DEPLOY	1
✓	POST_DEPLOY_VERIFICATION	duplocloud/zbuilder:v7	PHASE=POST_DEPLOY_VERIFICATION, FOO=BAR	2

Katkit Config

The above configuration customizations like Phases, ENV, etc. can be saved in the repository in a config file called katkitconfig.js Following is an example of one such file



```
[
          {
                "EnvName": "default",
                "LocalFleet": "true",
                "WorkFlow" : [
                                           {
                                                  "Name":"PRE_DEPLOY_BUILD",
                                                  "PhaseType":4,
                                                  "BuildParams":"PHASE=PRE_DEPLOY_BUILD, FOO=BAR",
                                                  "Order":0,
                                                  "Parallelism":1,
                                                  "ContainerImage":"duplocloud/zbuilder:v7"
                                           },
                                           {
                                                  "Name":"DEPLOY",
                                                  "PhaseType":1,
                                                  "BuildParams":"PHASE=DEPLOY",
                                                  "Order":1,
                                                  "Parallelism":1,
                                                  "ContainerImage":null
                                           }
                                        ]
          }
]
```

Advanced Functions

- Bring-your-own-image: By default, all tenant CI/CD runs are executed in a docker image specified by the administrator. This image would typically have the most common packages for your organization. But a user can bring his own builder image and specify the same. The image should have the Katkit agent that can be copied from the default builder image.

- Bring-your-own-fleet or Local Fleet: By default, Katkit will run the builder containers in a separate set of hosts, but the user can also choose to run the build container in the same tenant hosts which is being tested.