Azure 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.

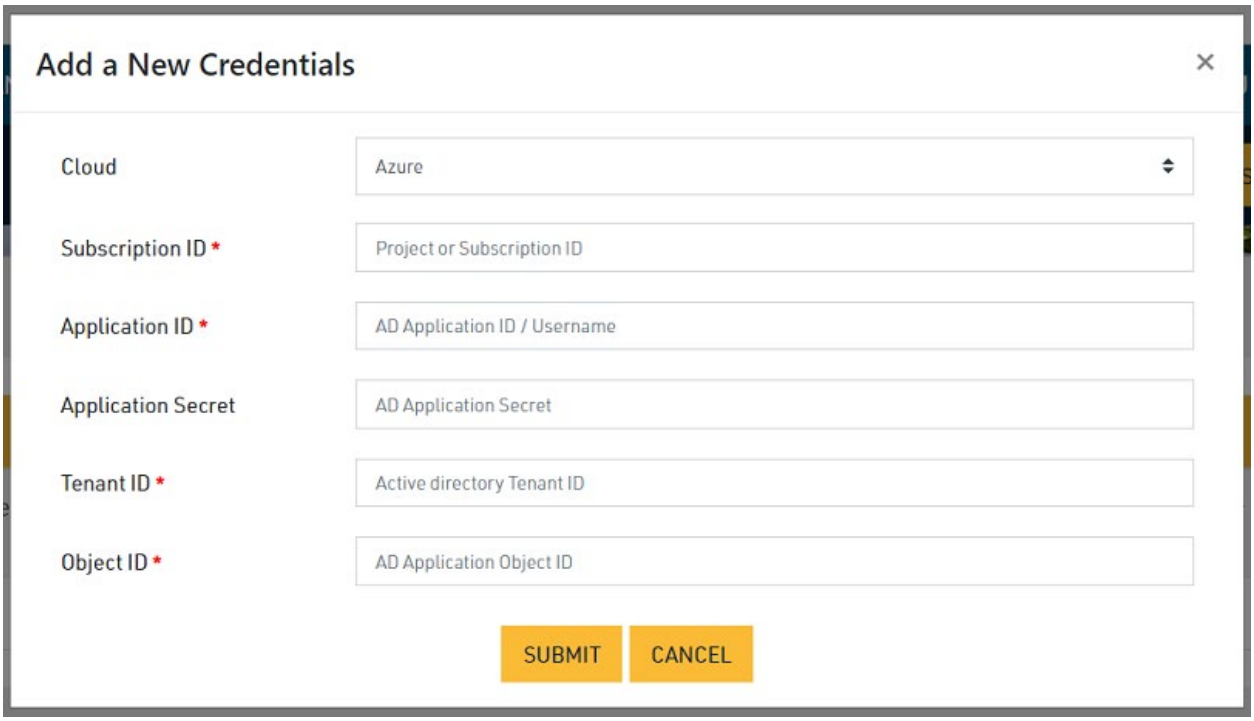
Terminologies

- **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.

Azure Guide

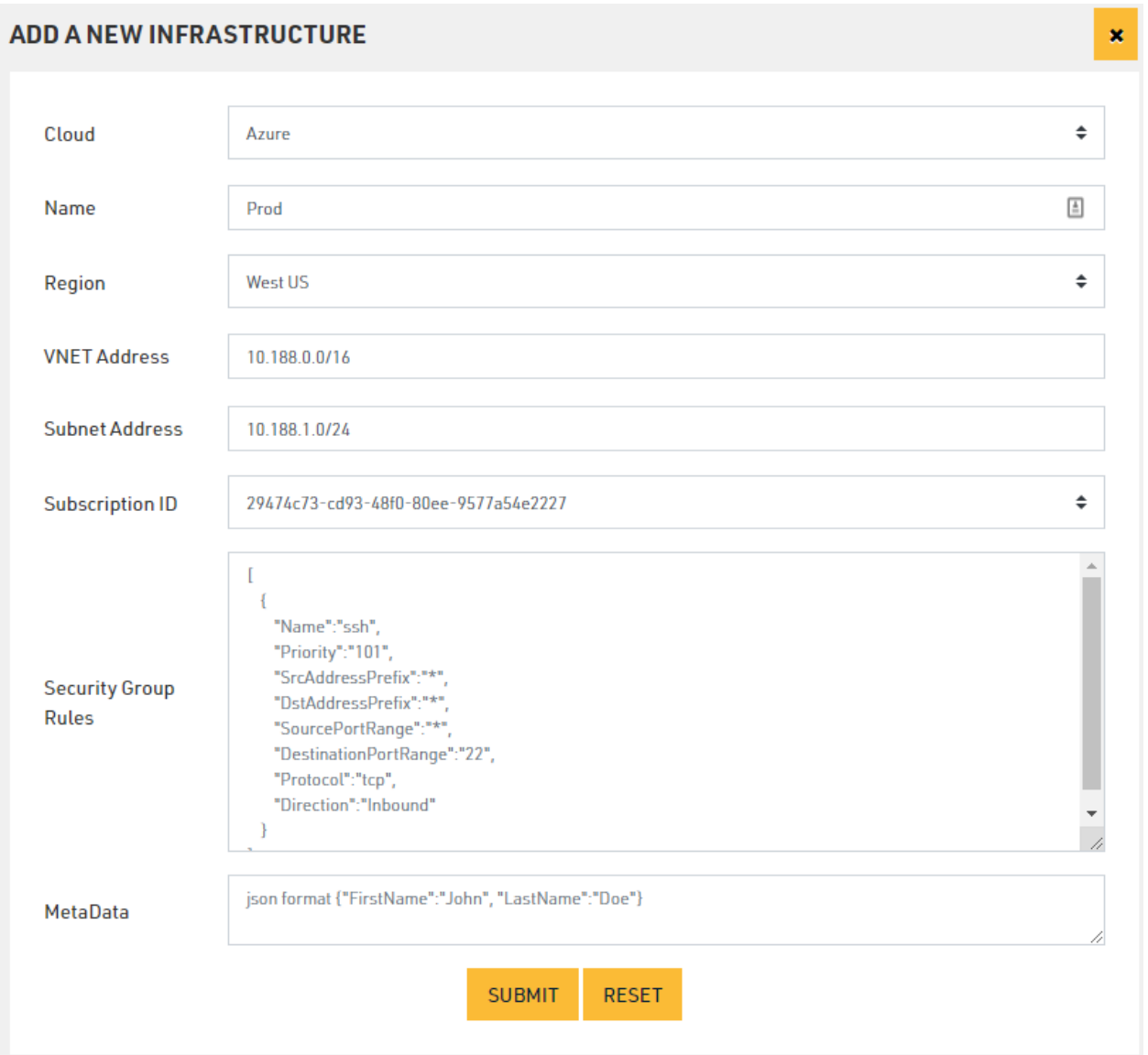
Subscription

The DuploCloud rules-based expert needs Azure Subscription details for it to Manage Cloud Resources. Proceed to **Admin → Credentials**. Click on **+button**. It opens a form where it asks to enter information about Azure Subscription ID, secrets.



Infrastructure

A completely new isolated environment can be created for <> from DuploCloud. Proceed to **Admin → Infrastructure**. Click on **+button**. It opens a form where it asks to enter basic information about the new environment that is needed.



DuploCloud will create a VNET with a default subnet and a default NSG. Creation of the Infrastructure takes around 10 mins. Once the Infrastructure is complete it comes to Ready state. A plan with the same name will be created for this Infrastructure. This plan can be used for creation of the tenants.

Key Vault

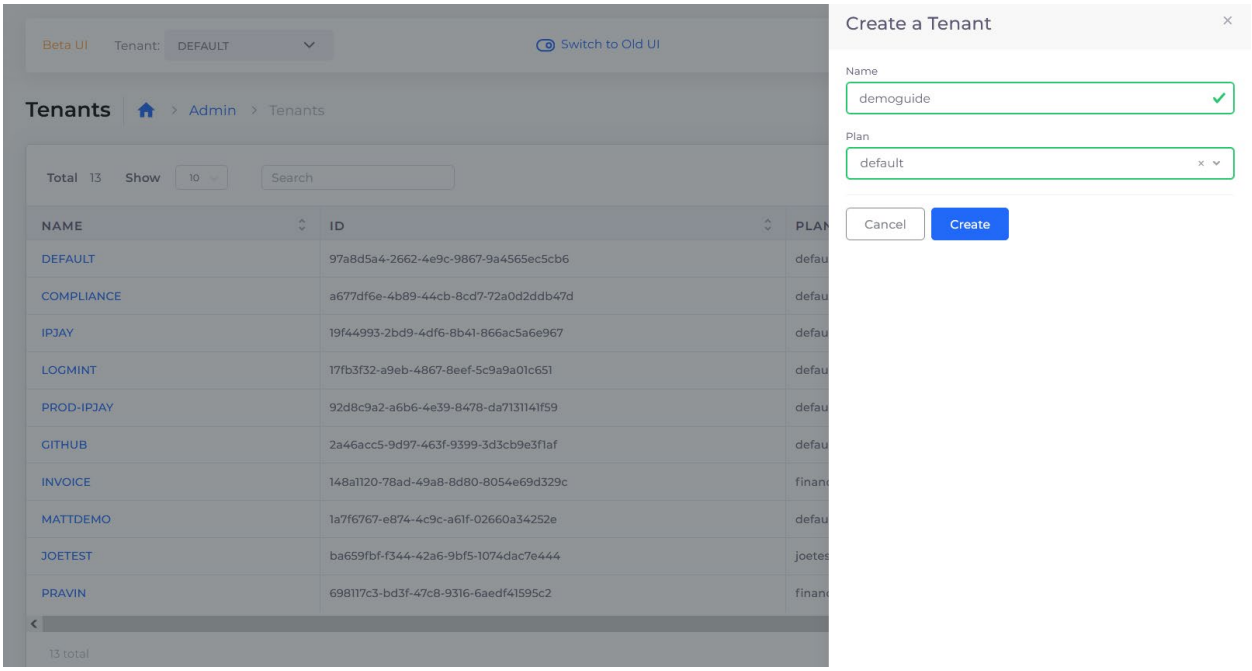
Documentation TBD. Please contact DuploCloud team for assistance.

Storage Account

Documentation TBD. Please contact DuploCloud team for assistance.

Tenant

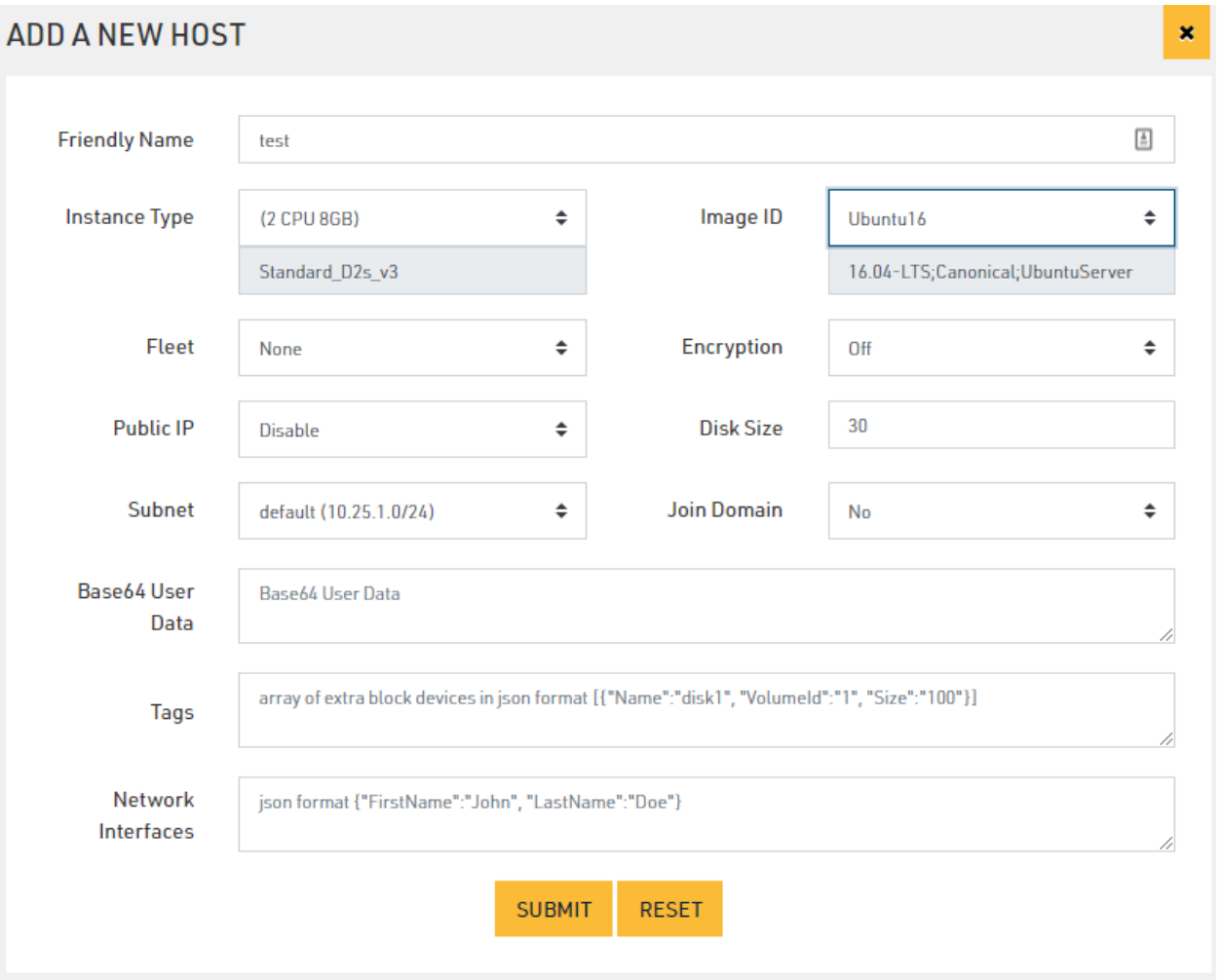
Create a new Tenant from the already created plan. Give a min for the DuploCloud to create the NSG for the tenant which acts as a security boundary.



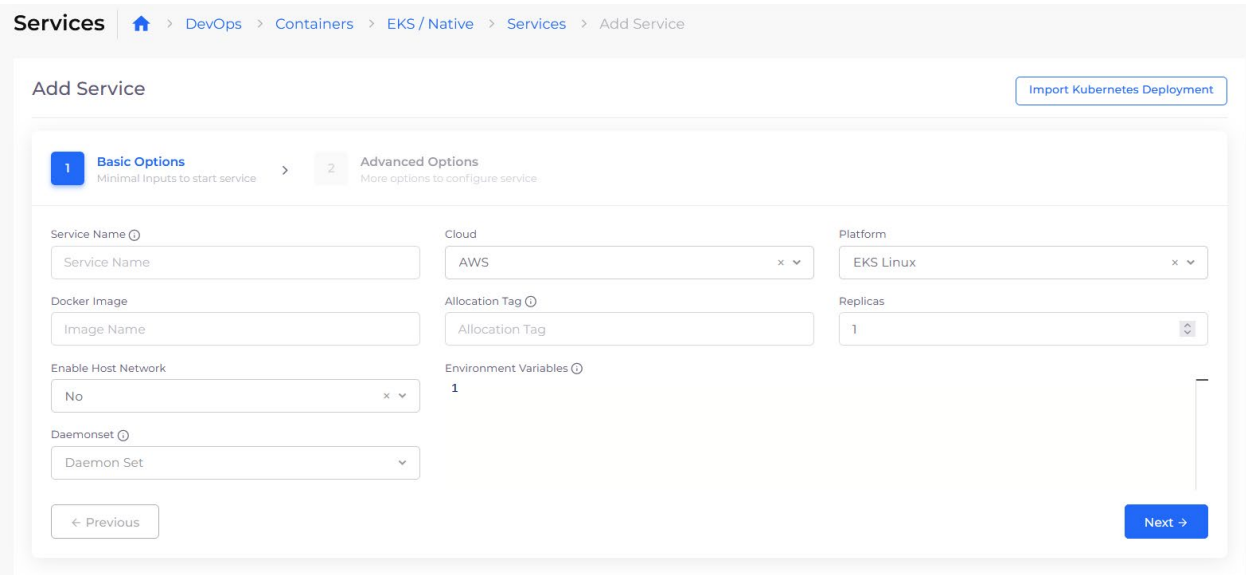
Quick Start

Switch to the Tenant we have created. Deployment is a three-step process: (1) Creating Host (VM), (2) Deploying a Service, (3) Exposing the service using LB.

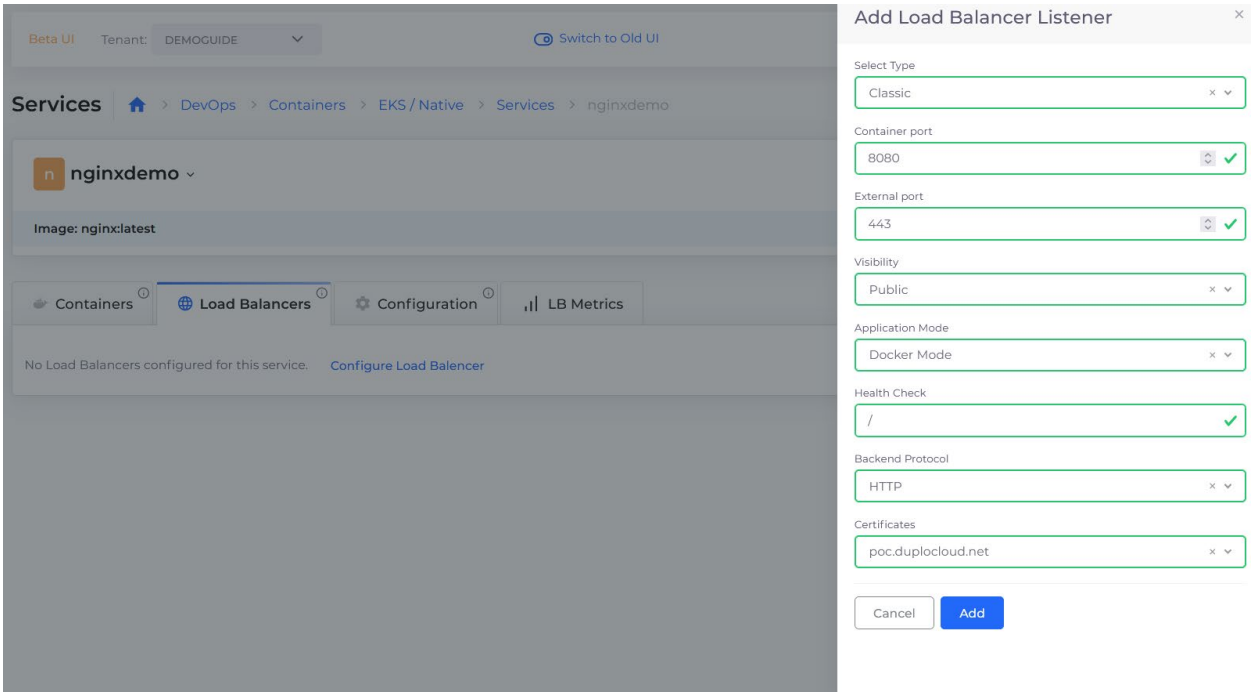
- **Create a Host (VM)** from the menu **DevOps → Hosts → Azure → Azure Hosts → + sign** above the table. Choose the desired instance type. The available instance types are set by your administrator or the plan you choose in DuploCloud. If you are not using this host for hosting containers, then set the pool as none. If you want a public IP for the host, then enable the public IP.



- **Deploy a Service (application)** from the menu **DevOps → Services → + Sign** above the table. Give a name for your services (no spaces); number of replicas; Dockerimage; volumes (if any). The number of replicas must be less than or equal to the number of hosts in the fleet.



- **Expose using LB:** Create an LB from the menu **DevOps → Services → + Sign** above the table for Load Balancer Configuration. The URL suffix you specify under Health Check will be used by DuploCloud during 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, then once it is running 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.



- 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 → Services → Select** the service from the table and click on edit.

Virtual Machine Scale Sets

Documentation TBD. Please contact DuploCloud team for assistance.

SQL Managed Instances

Documentation TBD. Please contact DuploCloud team for assistance.

SQL Database

Documentation TBD. Please contact DuploCloud team for assistance.

Containers

- **Built-In:** Refer to AWS Built-In
- **AKS:** Refer to K8 Deployment

Webapps

Documentation TBD. Please contact DuploCloud team for assistance.

Functions

Documentation TBD. Please contact DuploCloud team for assistance.

WAF

Documentation TBD. Please contact DuploCloud team for assistance.

Tables

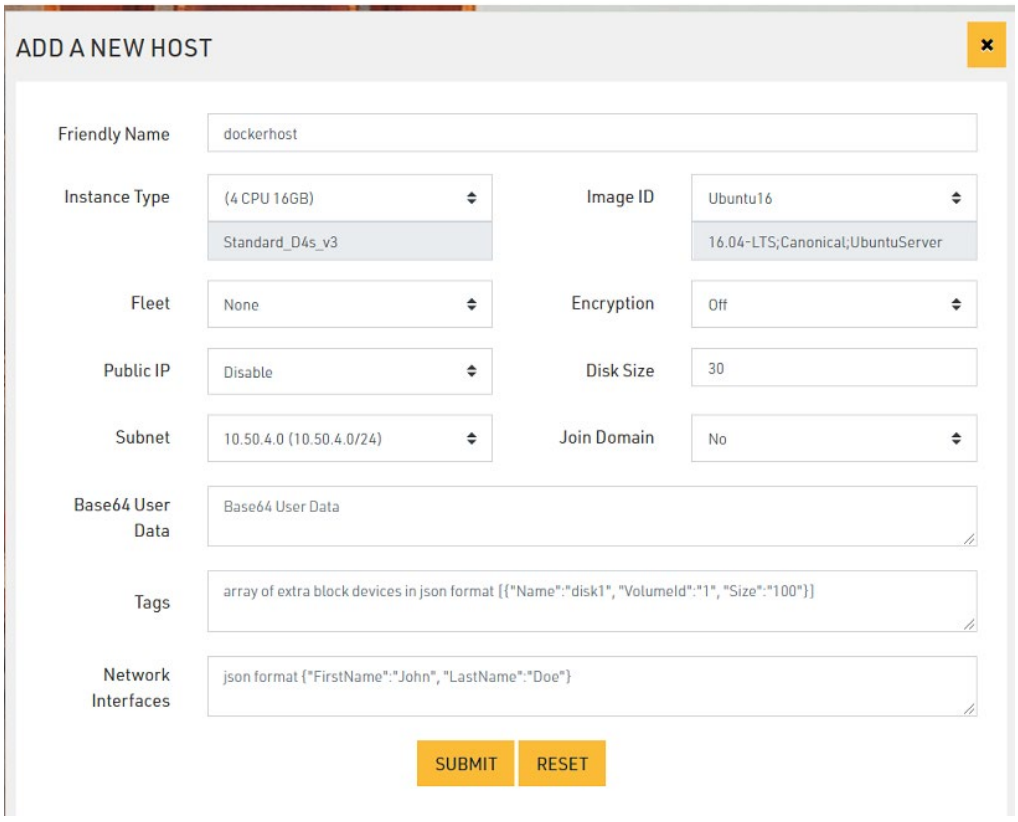
Documentation TBD. Please contact DuploCloud team for assistance.

Azure Use Cases

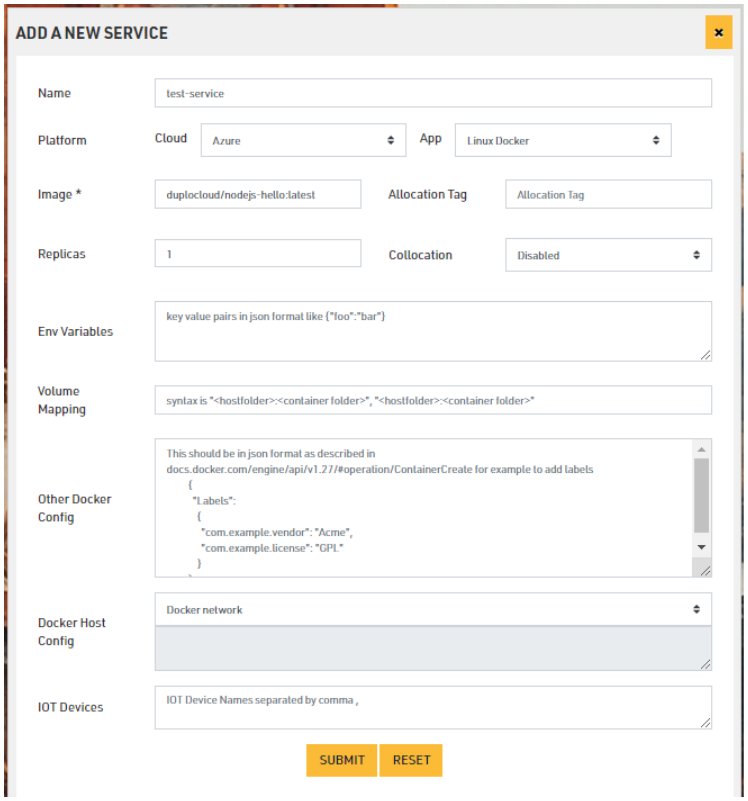
Docker Web App

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 DuploCloud's repository on Docker Hub.

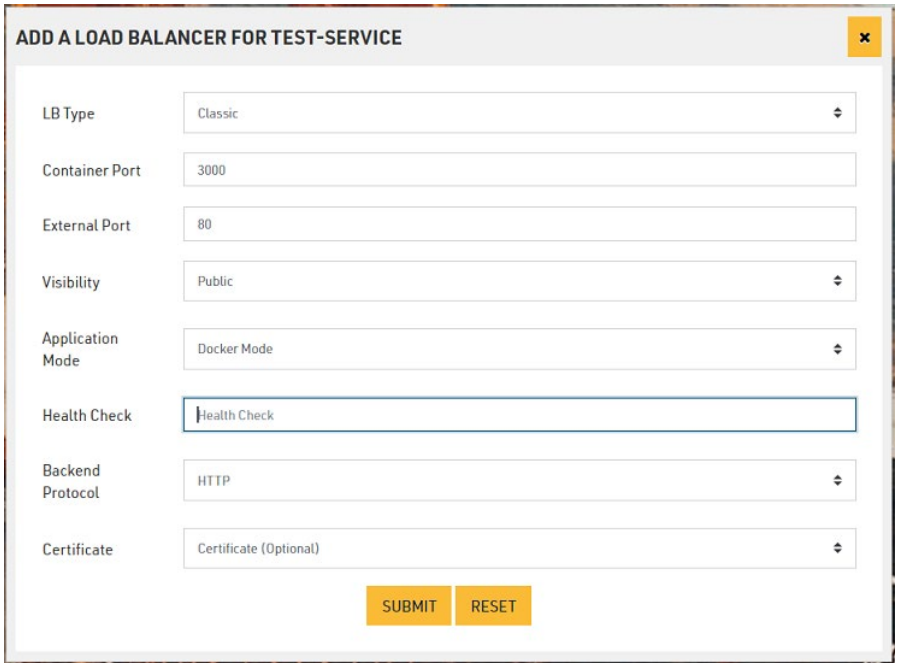
- Login to your DuploCloud console.
- Select "Deployments" from the tab on the top left corner.
- Select "Hosts" from the tabs. Select "Azure" Cloud on the left-side Section. A Host is the instance in which your Docker container will run. You should choose a host with appropriate processing capacity for your application.
- Click on the + icon to choose your host. Fill out the advanced configuration form if required and click submit.



- 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 the + icon 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. 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.



- Hit Submit! 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 the plus icon on the load balancer configuration table. Fill the menu as shown below and click submit.



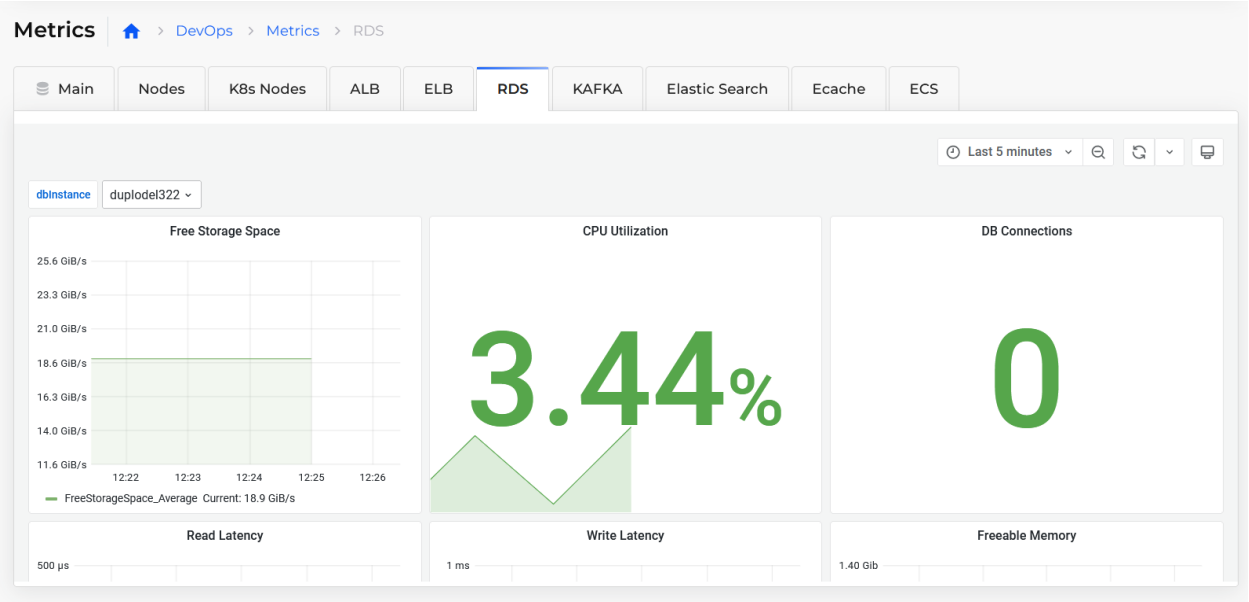
- Please wait for ~5 minutes as it can take a while for the Load Balancer to get provisioned.

Serverless Web Apps

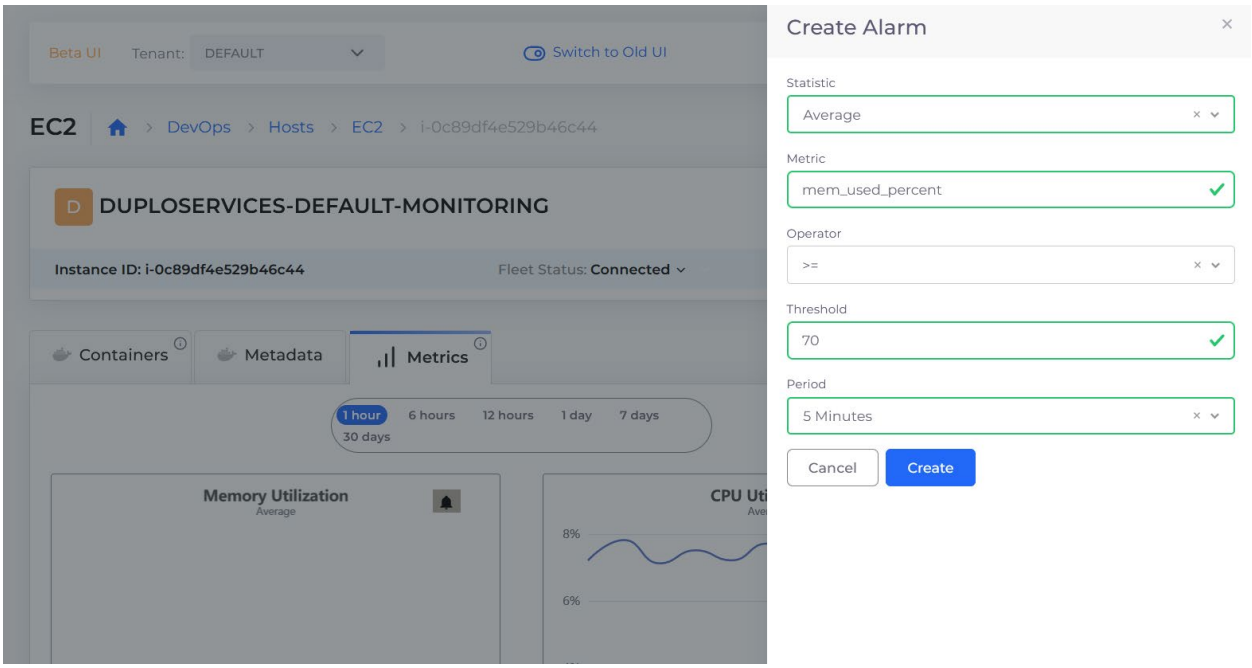
Documentation TBD. Please contact DuploCloud team for assistance.

Metrics

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

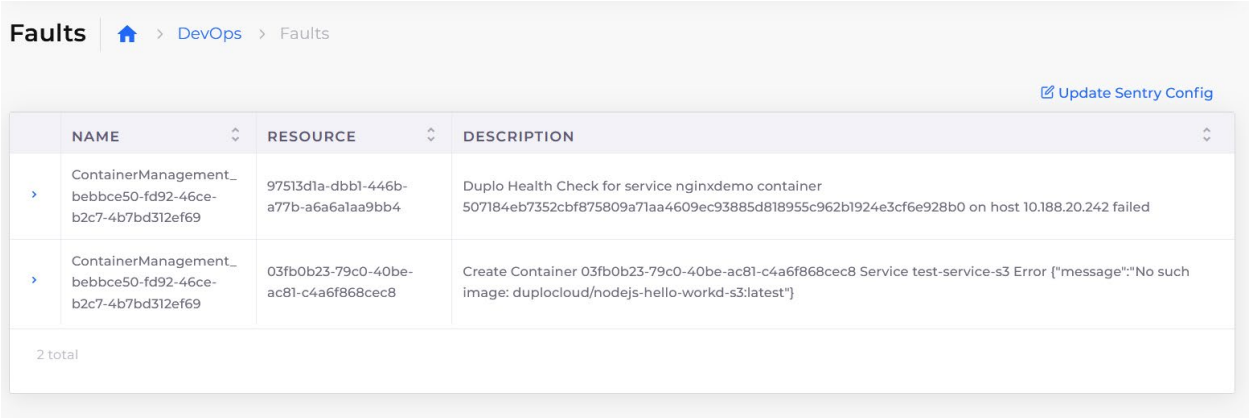


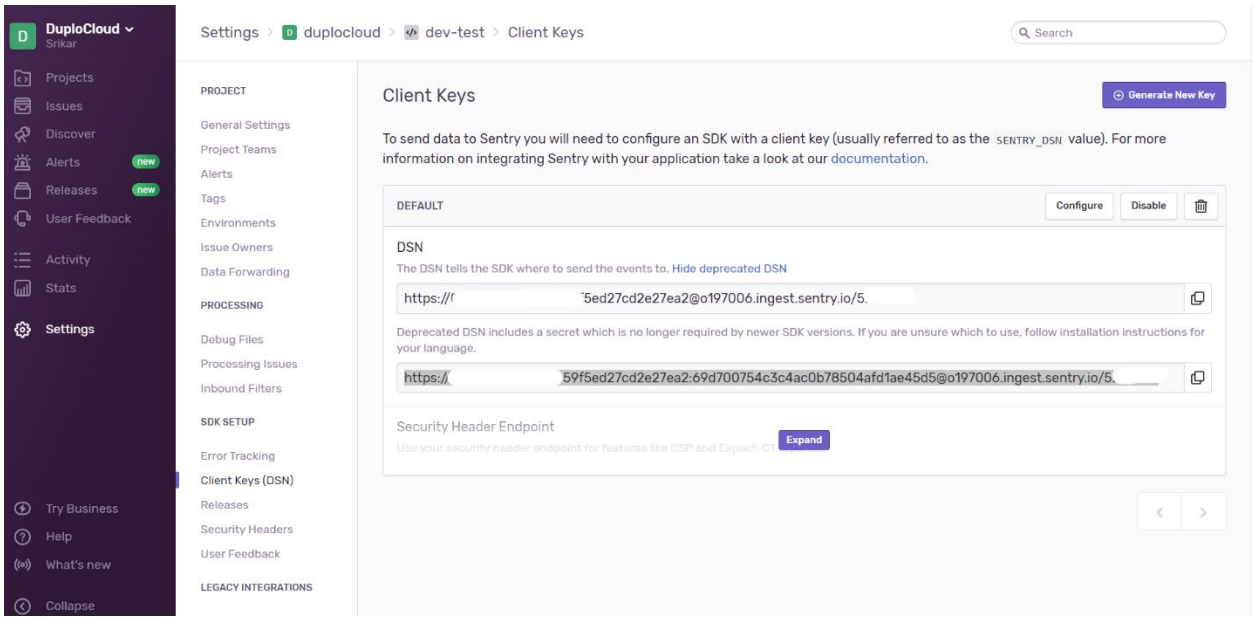
You can also set the AWS Alarms/Azure Alert rules for the individual metrics, click on the bell icon on any of the metrics. A form to create alarm shows up. You can provide the necessary information and create the alarm. If the AWS Alarm get triggered or any Azure alert is raised, DuploCloud will fetch it and show up in the faults. Refer Faults/Alerting section for more information.

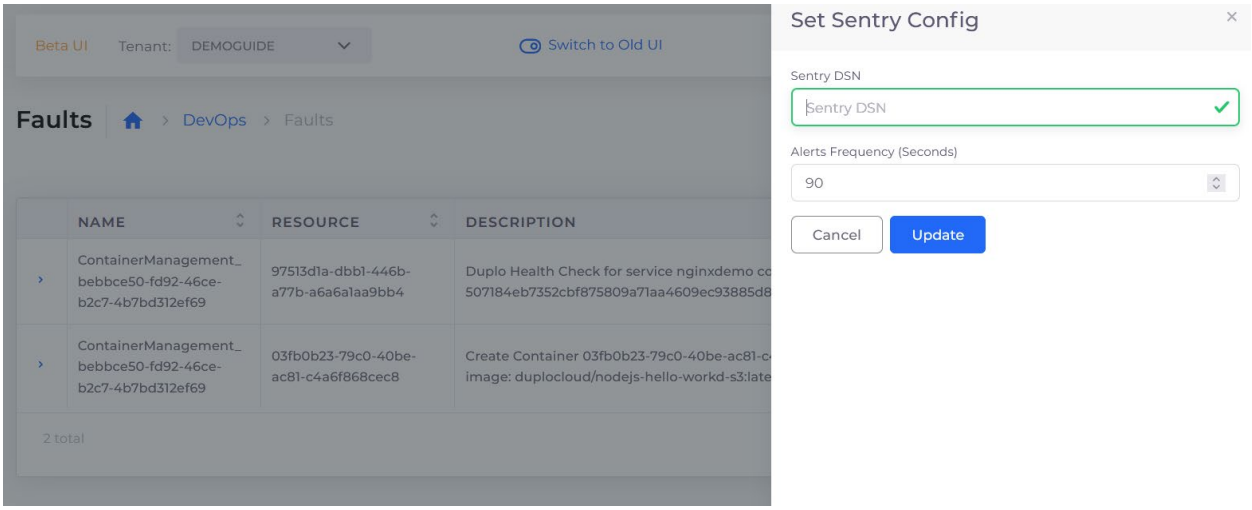


Faults/Alerting

Faults that happen in the system be it Infrastructure creation, Container deployments or Application health checks can be tracked in the DuploCloud portal under Faults Menu. You can look at Tenant specific faults under **Deployments → Faults** or all the faults in the system under **Admin → Faults**. In addition to notifying you about the faults, DuploCloud integrates with Sentry, which will send an Email alert for the fault and also 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 **Deployments → Metrics → Sentry DSN**.

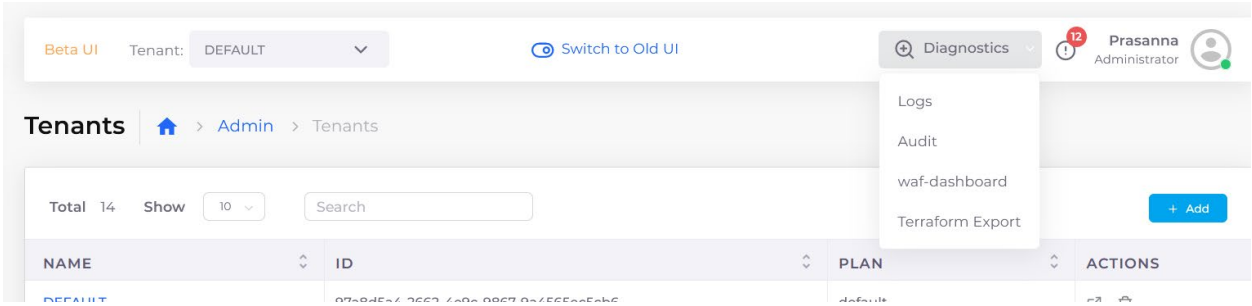




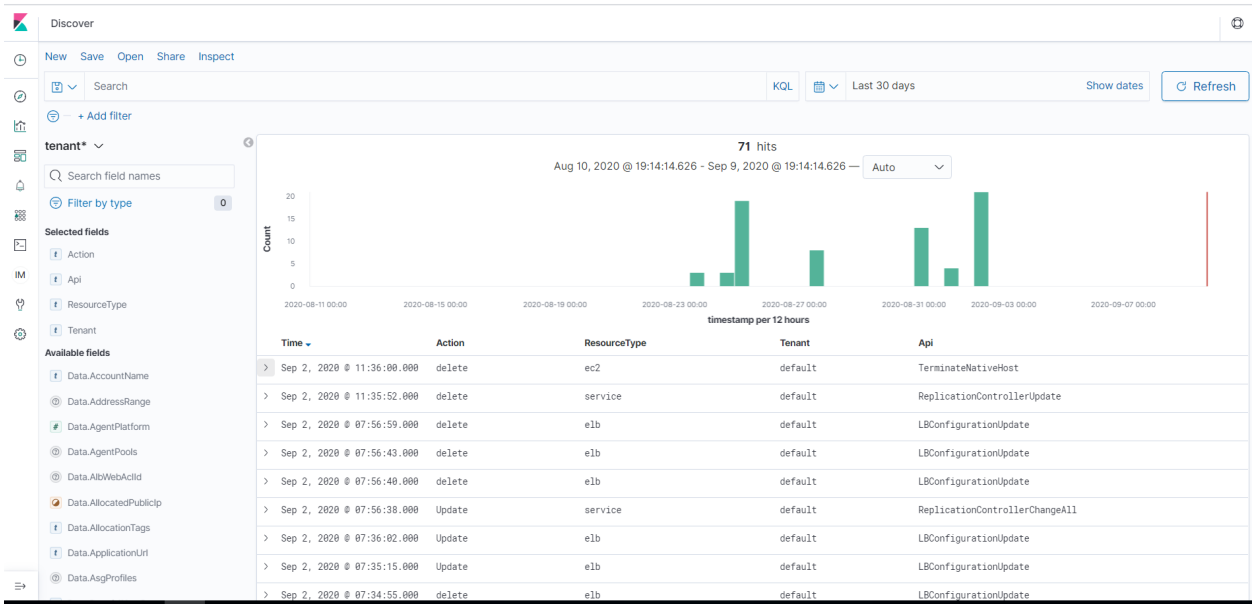


Logs

All the activity in the DuploCloud is logged which can be used for auditing. All the logs are saved into ES 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.



Backups

Backup of the resources be it DB Snaphsots, VM Snaphsots/VM AMI can be done directly under the resource in the DuploCloud.

Disaster recovery

Documentation TBD. Please contact DuploCloud team for assistance.

Security

Documentation TBD. Please contact DuploCloud team for assistance.

Cloud Migration

Documentation TBD. Please contact DuploCloud team for assistance.

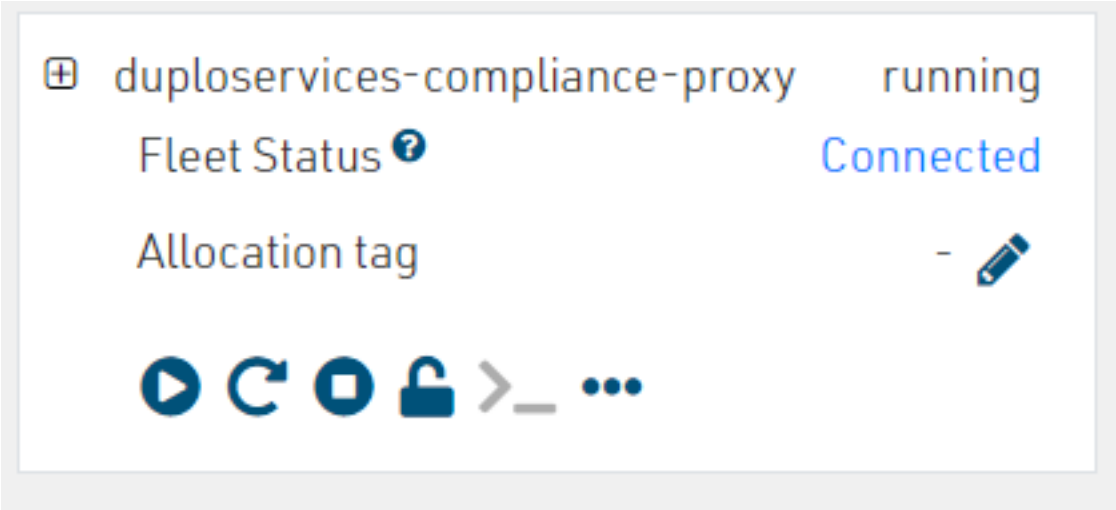
Remote Working

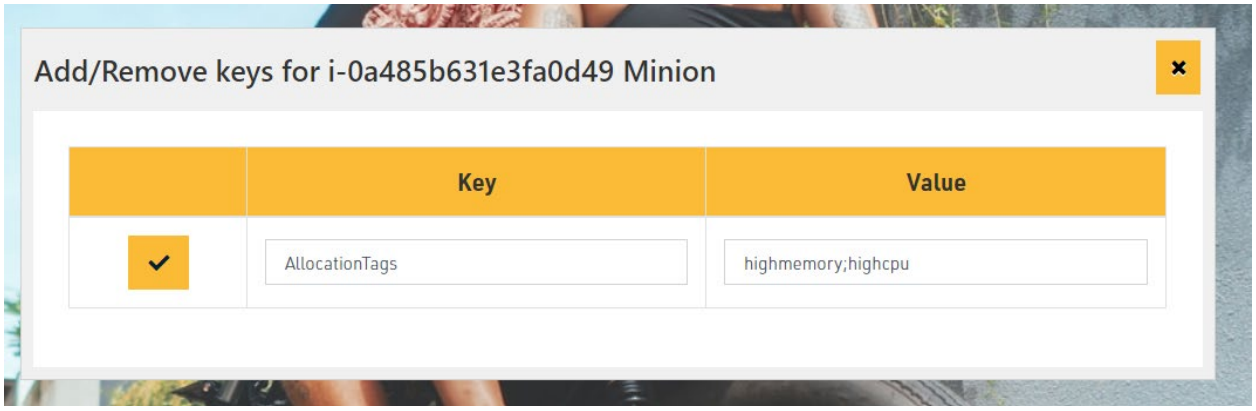
Documentation TBD. Please contact DuploCloud team for assistance.

Additional Deployment functionalities

Advanced Functions

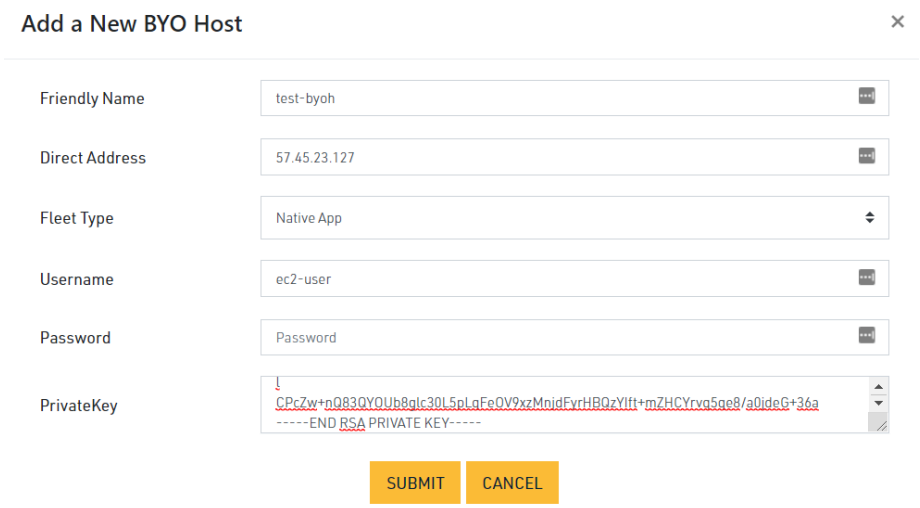
- **Allocation tags:** By default, DuploCloud spreads the container replicas across the hosts. By virtue of allocation tag the user can pin a container to a set of hosts that have a specific tag. Click on the edit icon next to allocation tag under host. "`AllocationTags`" as key is already pre-populated,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

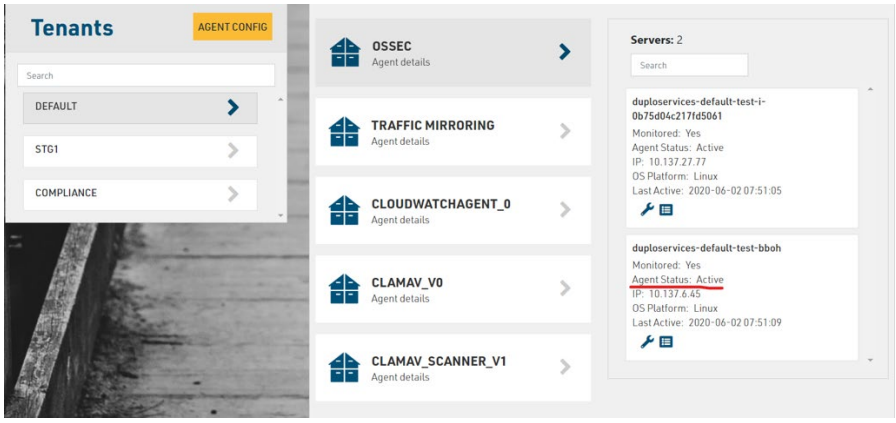




BYOH

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.





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

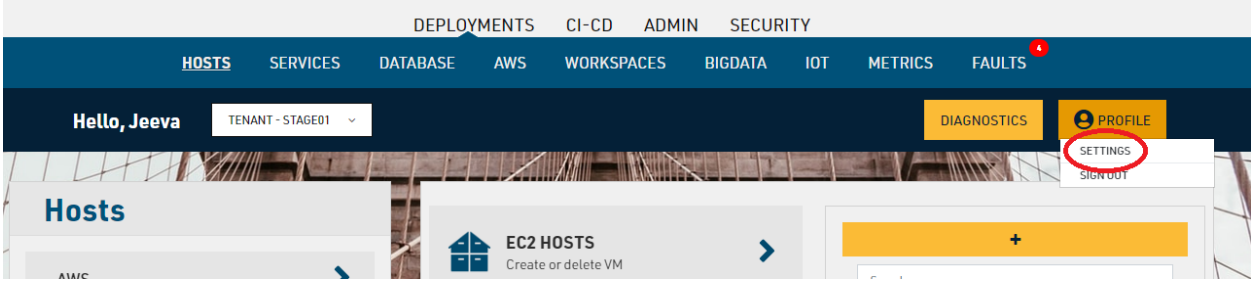
There are 2 types of roles, user and administrator. Each user can have access to one or more tenants. Each tenant can be accessed by one or more users. Administrator has access to all tenants plus the administrative functions like plan configuration, system dashboard, system faults, etc.

Accessing Cloud Server

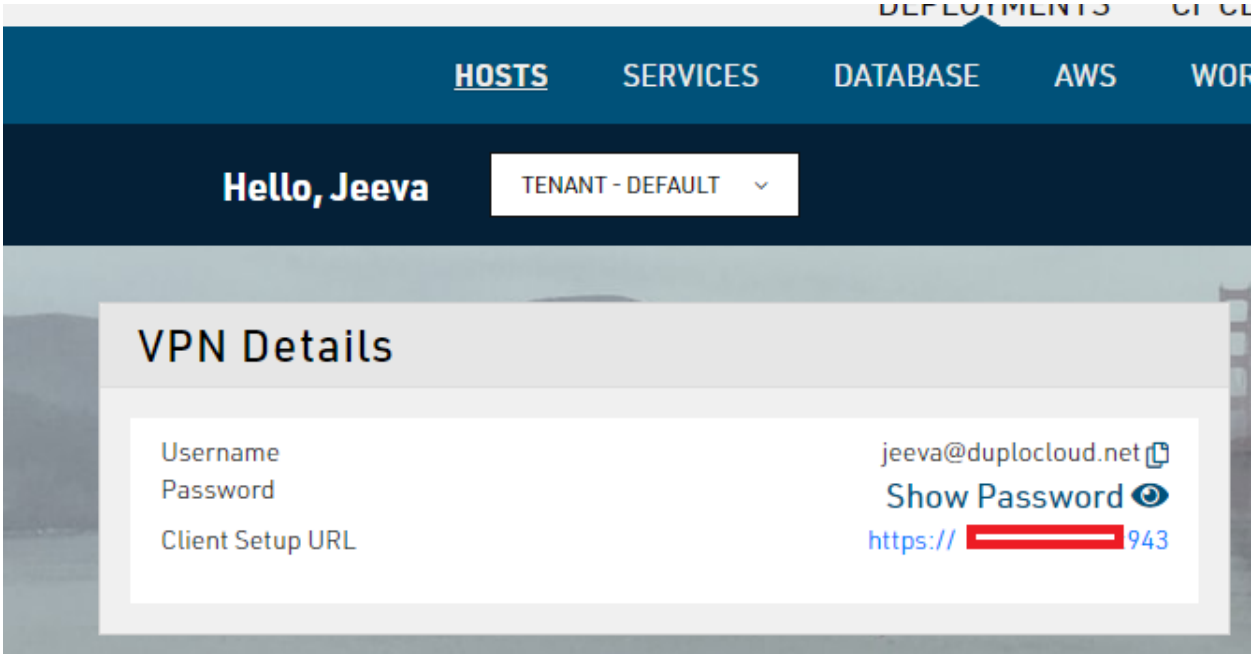
VPN Access

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.

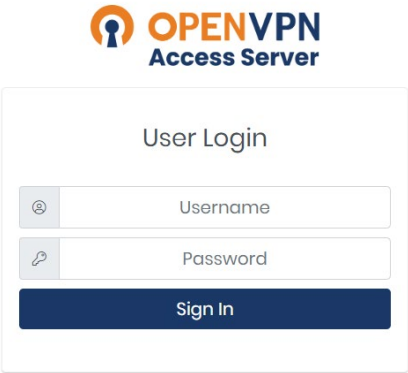
- **Setup OpenVPN client**
  Login to OpenVPN web portal to download OpenVPN config and client.
- Click on **Profile → Settings** as shown in image below



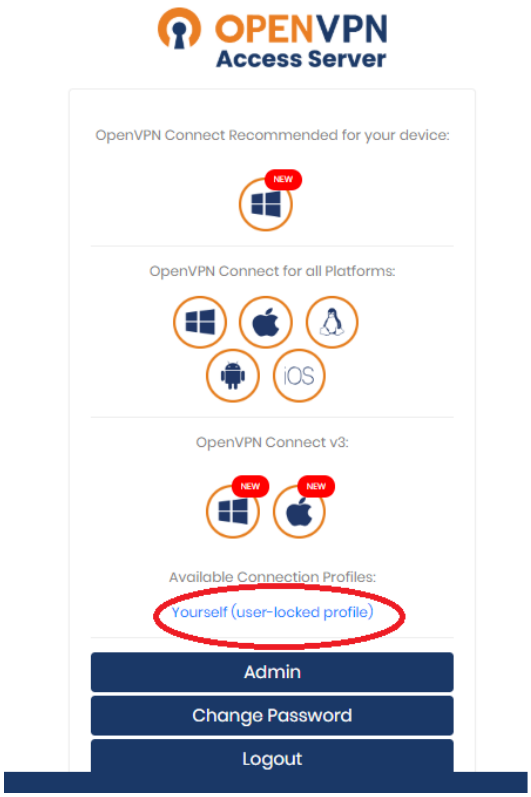
- Click the openvpn link which will open a new tab.



- Login using Username and Password provided.



- Download the openvpn config and client.

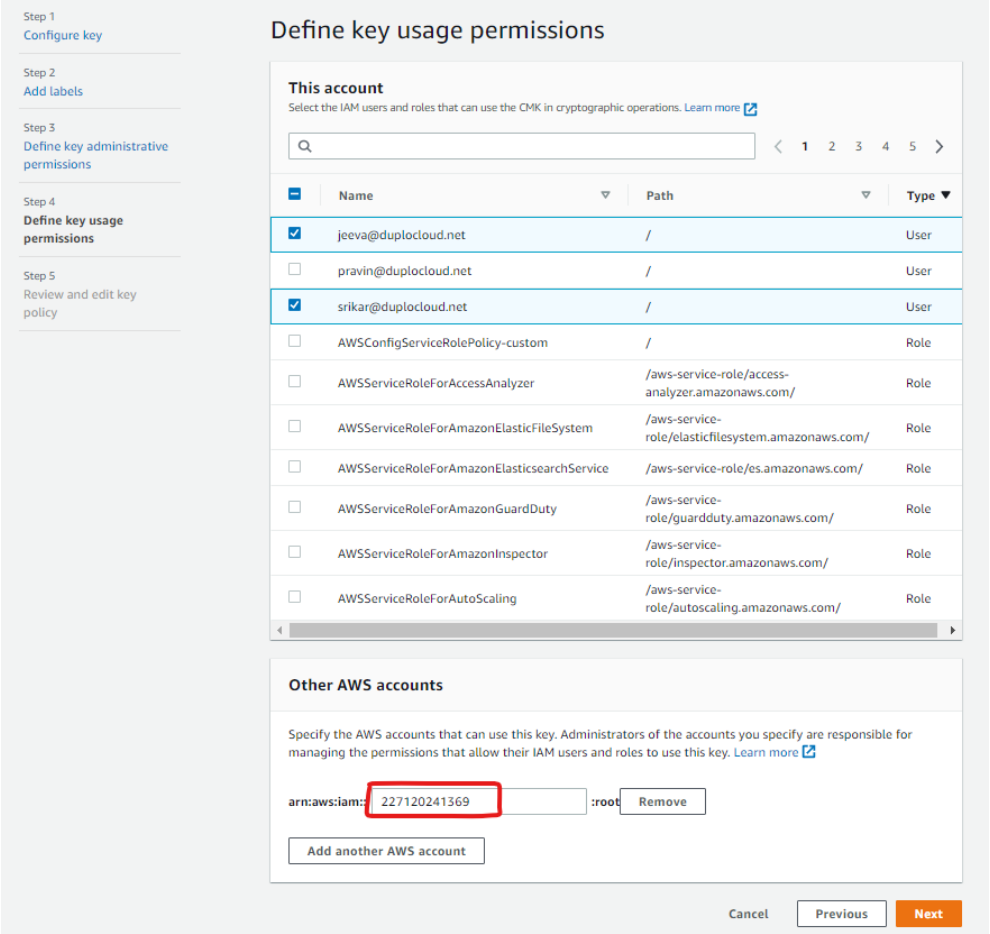


- Now you are ready to connect with VPN using OpenVPN client.

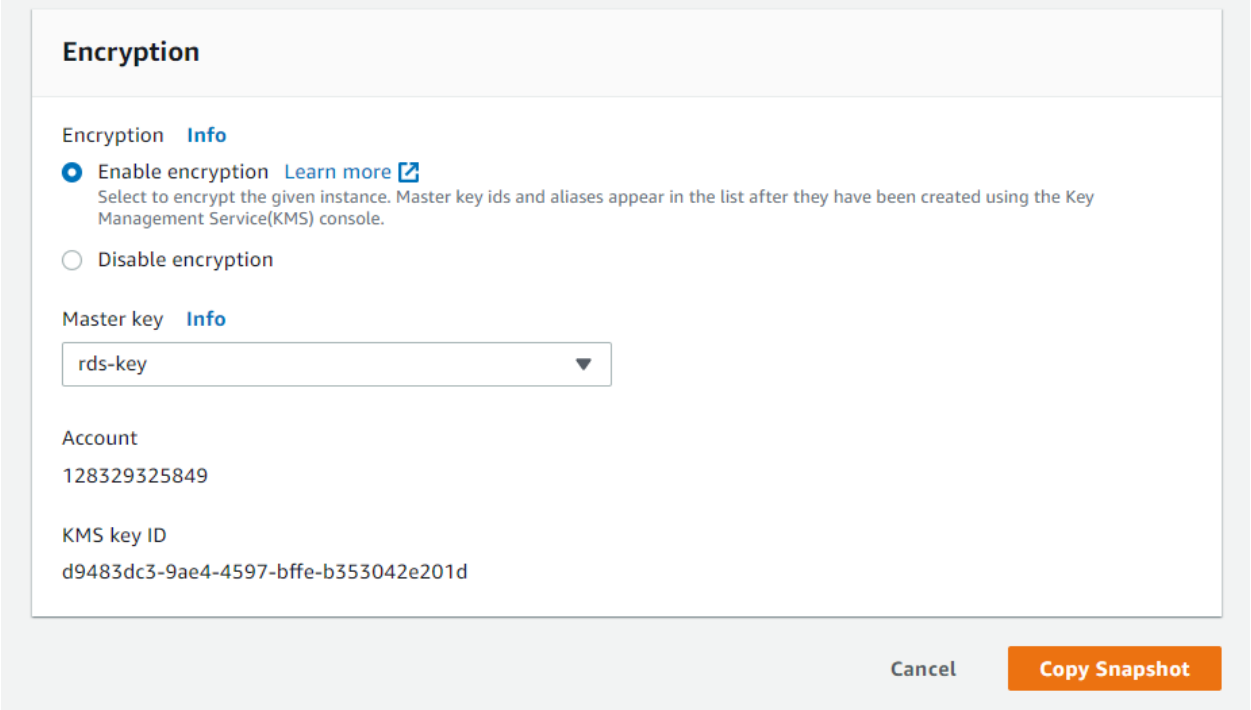
Sharing Encrypted DB

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.

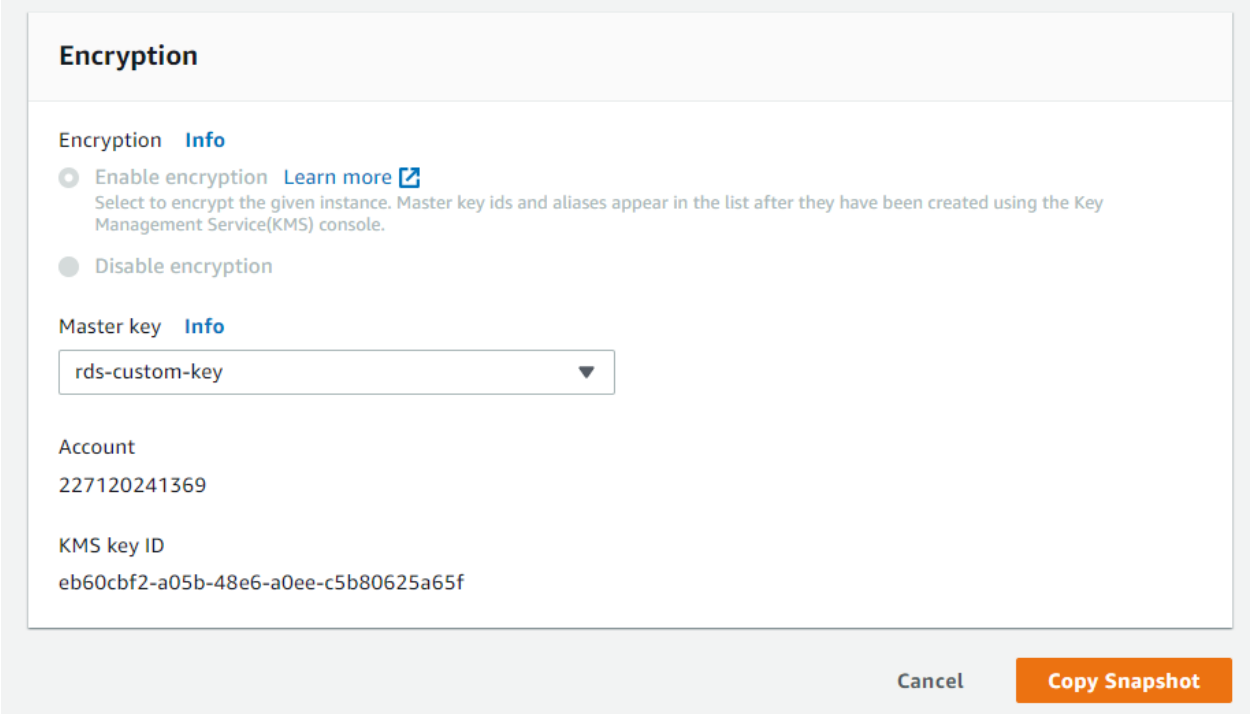


- Once the key is created, go to **RDS → Snapshots**, select the snapshot and click Copy Snapshot. In the encryption change the master key to the key we created before.

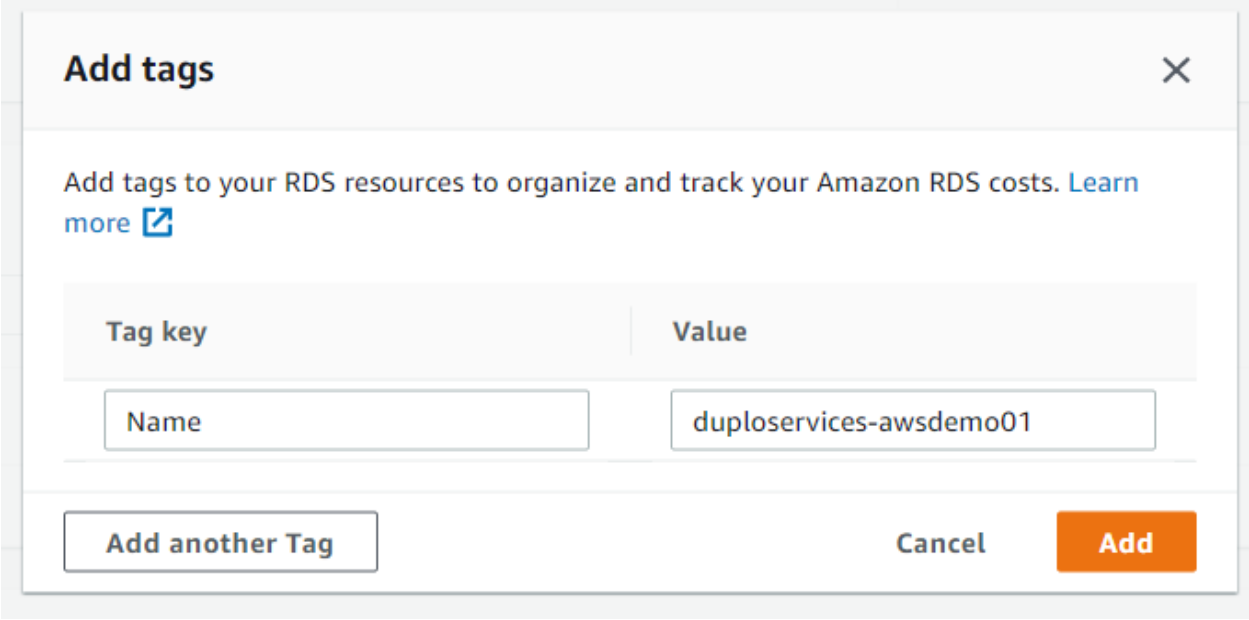


- Once the copied snapshot is ready, as usual share the snapshot to another account by clicking share snapshot and providing the other account id.

- 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.

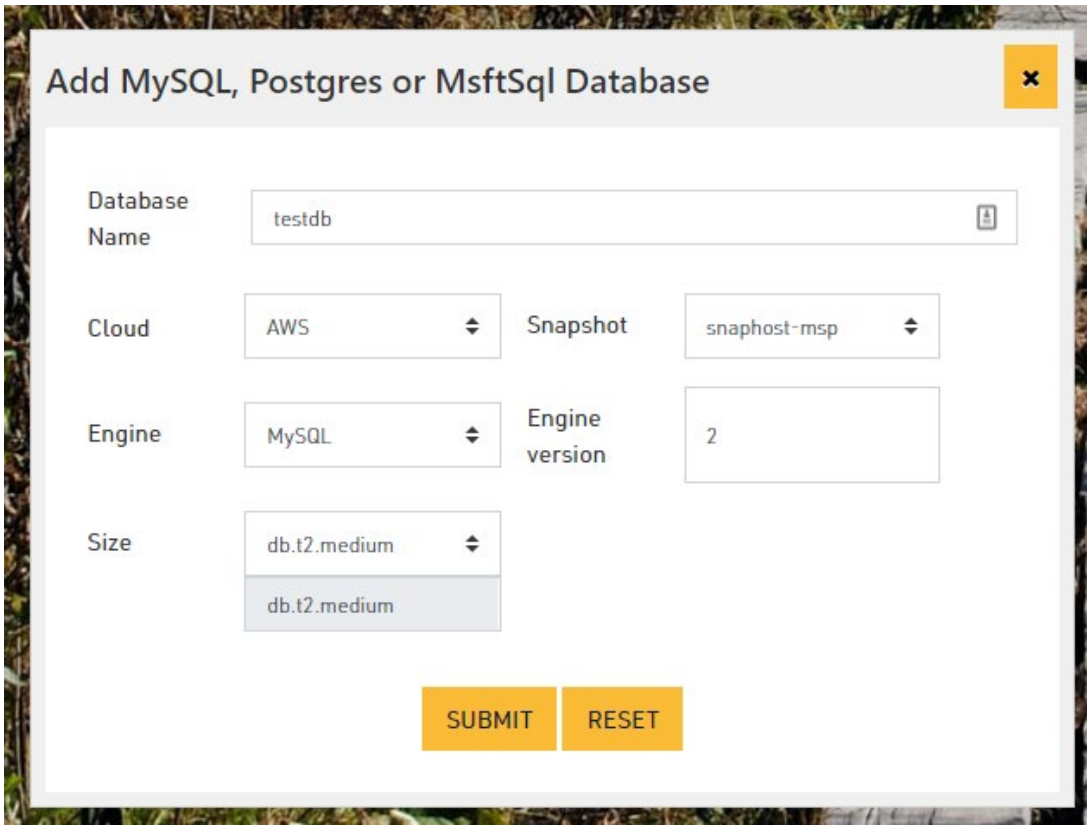


- In the copied snapshot add a tag with Key as "Name" and Value as "duploservices-{tenantname}" where tenantname is the tenant where u want to launch an RDS with this snapshot.



- 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

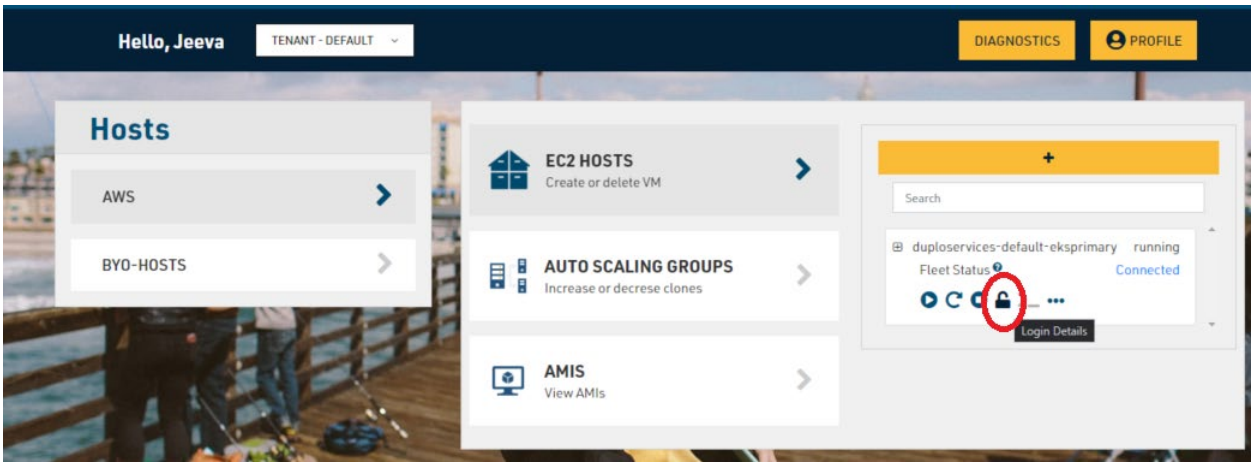
type and hit submit. In few mins, the DB will be created with the data from the snapshot. You must use the existing username and password to access the DB



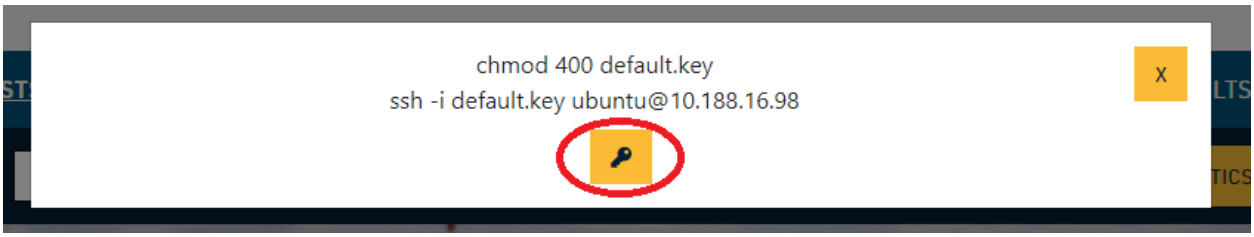
SSH EC2 Instance

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.



- 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 servicdescription.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.

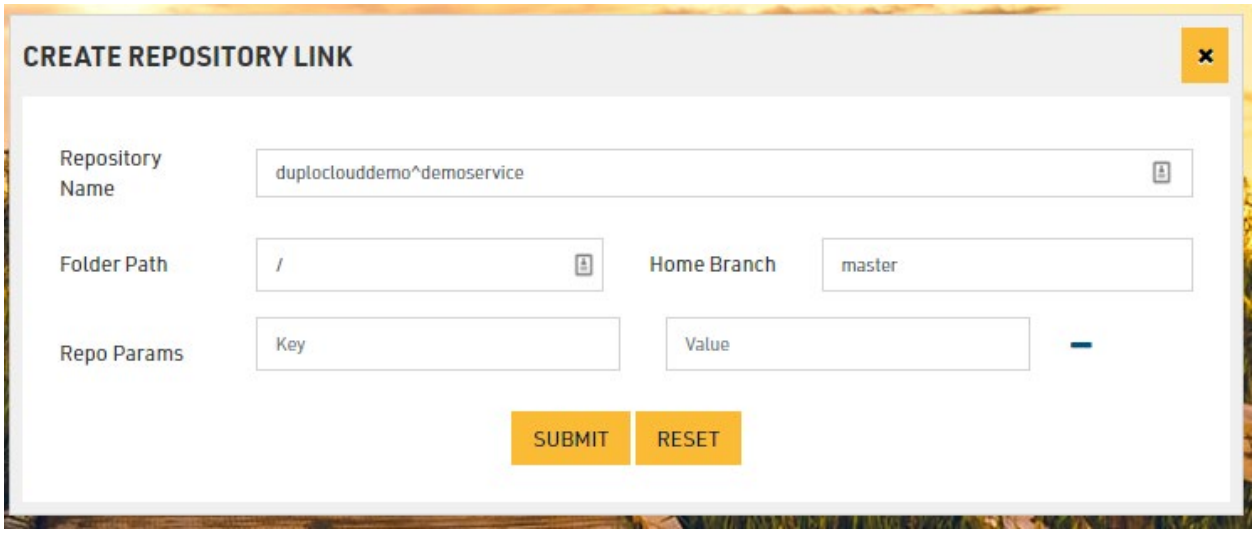
Export Service Description

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

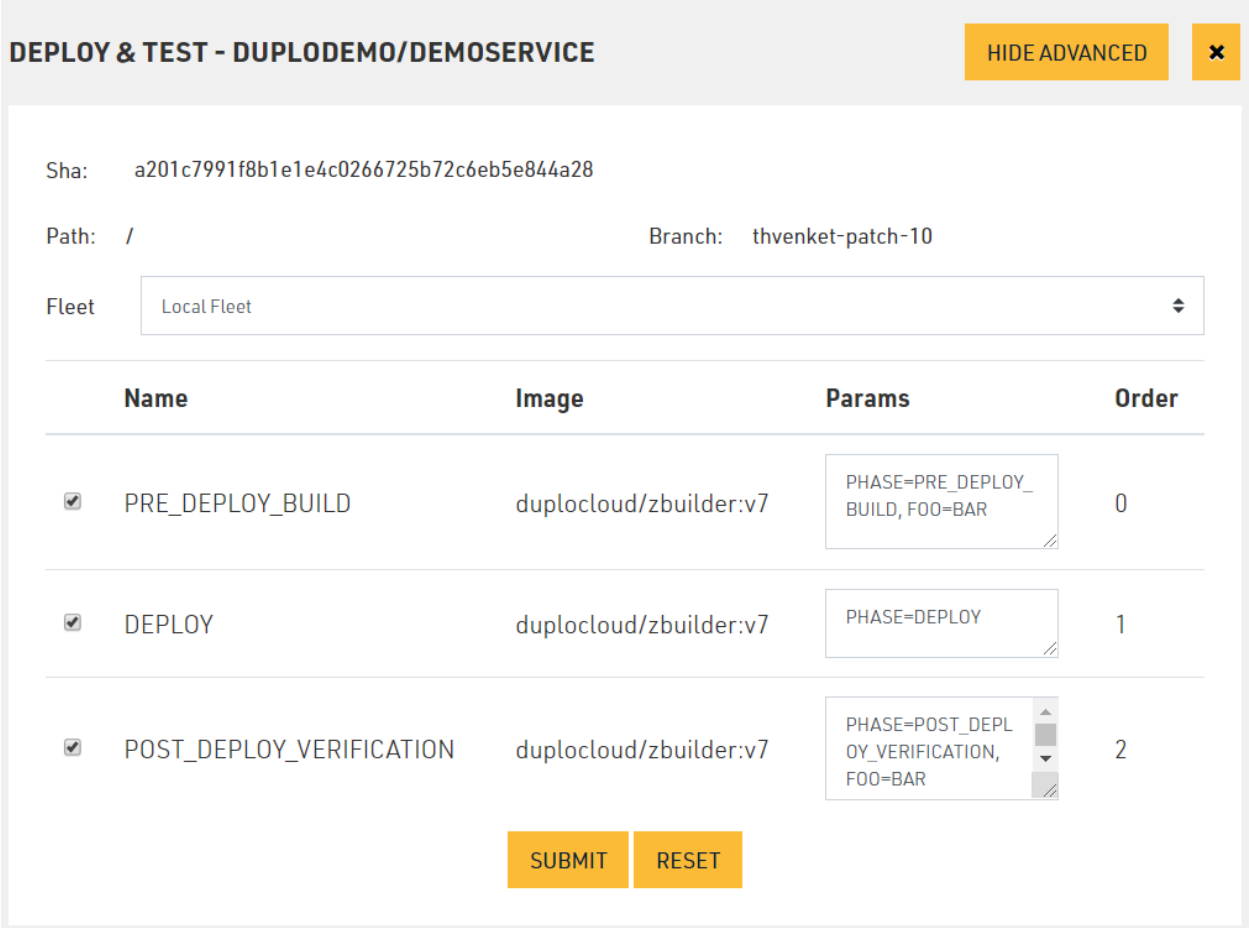
Once the above steps have been performed, you can link your GitHub Repository to your tenant. In addition to the repository name, you also need to specify the "Home Branch" which is the branch for which the PRs will be monitored by Katkit for the user to run deployments. Same repository and branch combination 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.



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.



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.