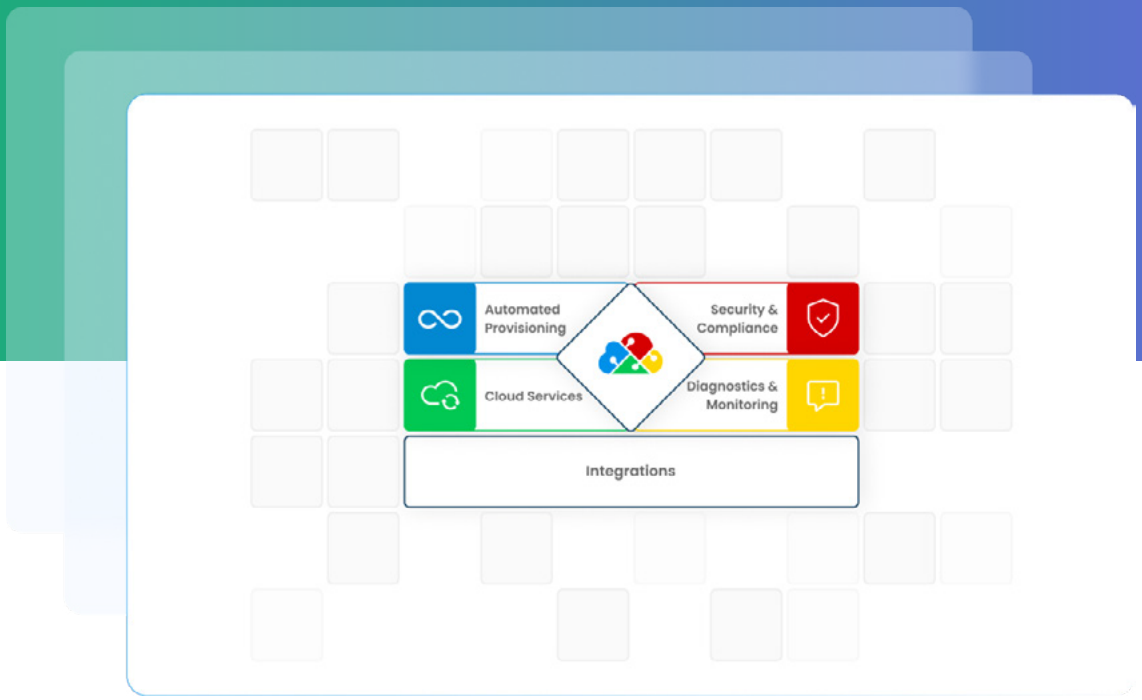
DuploCloud

WHITEPAPER

Accelerate Cloud Infrastructure Provisioning with DevOps Automation



Contents

Introduction

Many off-the-shelf components and services allow developers to create increasingly complex applications that can be scaled either on-premises or in the cloud. While this supplies greater flexibility and agility in terms of application development, the proliferation of these components and services has also caused a drastic spike in fragmentation throughout the infrastructure.

Applications that were formerly built with a few VMs to manage storage, computing, and networking resources are multiplying into scores of configurations that contain security groups, containers, namespaces, clusters, IAM roles, policies, object stores, NoSQL databases, and so forth. Building a fully automated and compliant infrastructure for even a 50-VM application in a regulated industry is a multi-month arduous process requiring uniquely skilled—and scarce—DevOps resources.

Enter **The DuploCloud DevOps Automation Platform**, a Low-Code/No-Code automation solution for DevOps that speeds cloud infrastructure provisioning by 10x while lowering costs by 75%.

This whitepaper highlights the typical approach toward DevOps using Infrastructure-as-Code and the associated challenges, alongside DuploCloud's approach to DevOps—an end-to-end platform that translates high-level application specifications into detailed cloud configurations while incorporating best practices around security, availability, and compliance guidelines.

The hyper-scale automation techniques described in this paper have been used inside cloud providers such as AWS and Azure for years, where just a thousand engineers are operating millions of workloads globally with top-notch availability, scale, security, and compliance standards. The DuploCloud team is among the original inventors of these automation techniques in the public cloud and is now democratizing this cost-saving, performance-boosting methodology for mainstream IT.

Watch a video of DevOps Automation by DuploCloud.

The Trend Towards Modern Cloud-Based Deployments

Three major shifts are happening across all industries today:

- Infrastructure is moving to the cloud and becoming 100% software-driven (Infrastructure-as-Code).
- Applications are getting more fragmented and diverse (microservices).
- An ever-increasing number of application functionality has moved out of the developer's code and into Platform Services, becoming completely managed by cloud providers.

With the increasing adoption of public clouds, enterprises want cloud operations to be 100% software driven. Applications are morphing into a combination of microservices using containers, managed services for databases, messaging, key-value stores, NoSQL stores, Lambda functions, and object stores.

To reach this 100% software-driven benchmark, the first step is creating a maintainable, scalable application blueprint.

Create an Application Blueprint

The first step towards the realization of cloud deployment is to draw out a high-level application architecture. This would typically be done by an architect in the organization. An example blueprint is shown in Figure 1, which depicts a deployment architecture for an application in AWS.

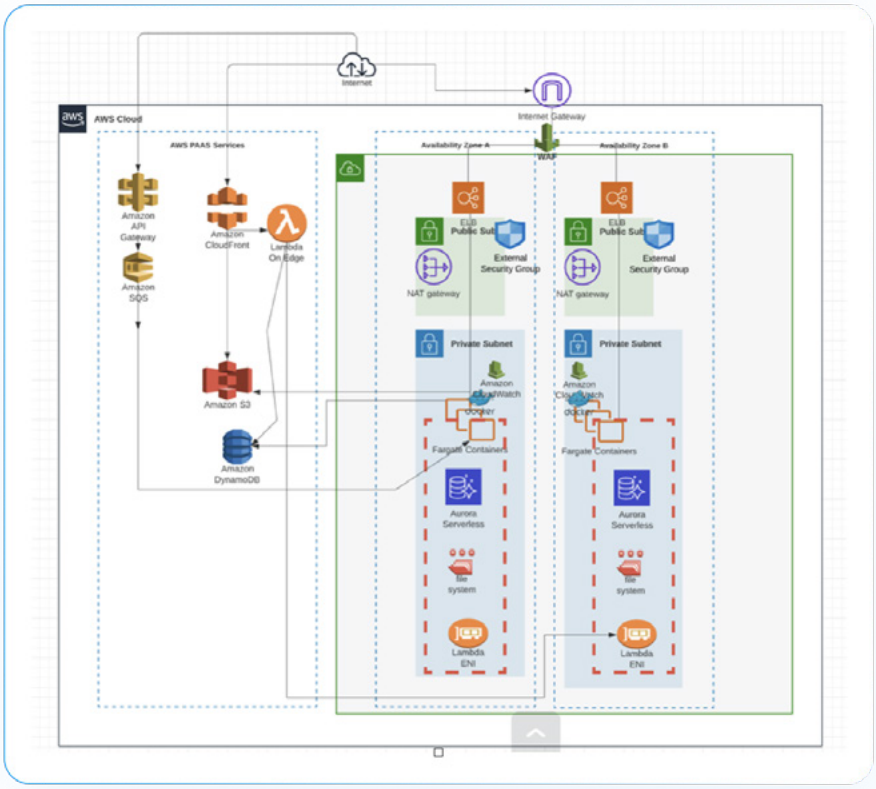


Figure 1

The topology consists of a set of microservices, packaged as Docker containers, running in AWS ECS Fargate. Aurora MySQL is used as a Database and services are exposed to the internet via an external load balancer, fronted by a Web Application Firewall (WAF). Another set of microservices can use Lambda functions and an API gateway. Data stores include S3 and DynamoDB, in this example.

However, if an organization is in Azure, the deployment architecture is Azure-specific. The constructs and terminology are conceptually the same as AWS. One such topology is shown in Figure 2.

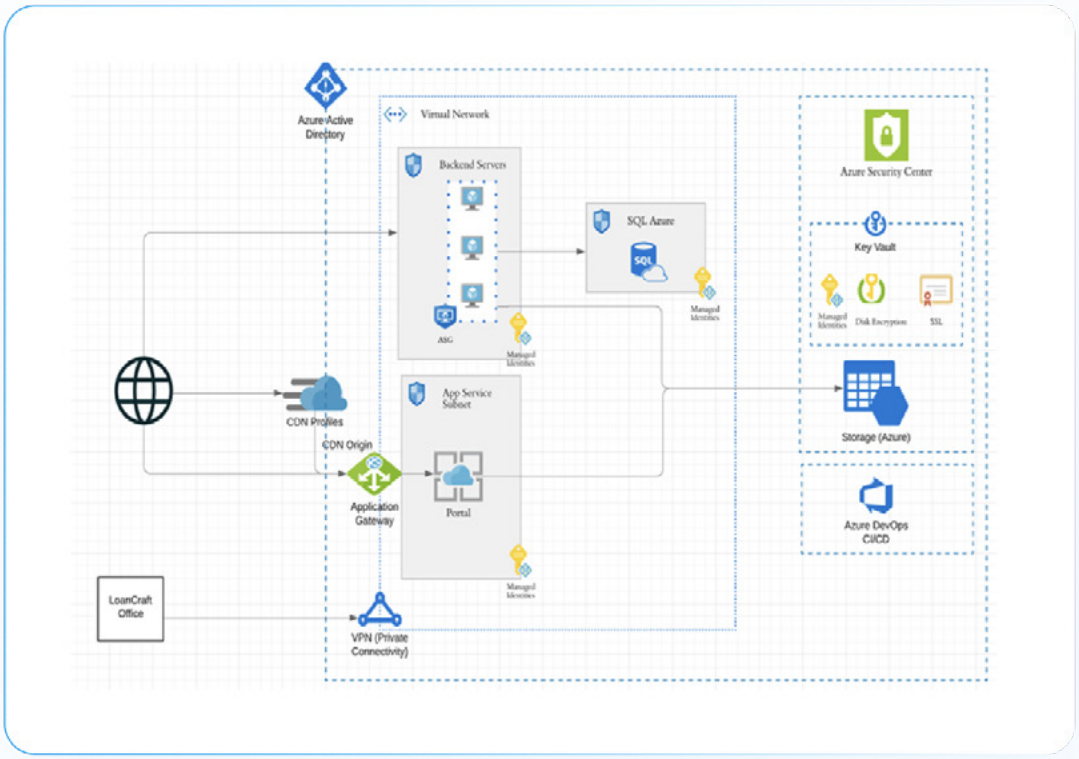


Figure 2

Key Pain Point: The high-level architecture gets passed to DevOps teams, translating it into hundreds of lower-level cloud configurations requiring thousands of lines of Infrastructure- as-Code. Deep subject matter expertise is needed both in operations and programming, which is a hard-to-find skill! Ever heard of a sysadmin who loves Java or a .NET developer who knows the nuances of security best practices?

DevOps Lifecycle

The infrastructure configuration to realize application blueprints is typically done in a series of phases as shown in Figure 3.

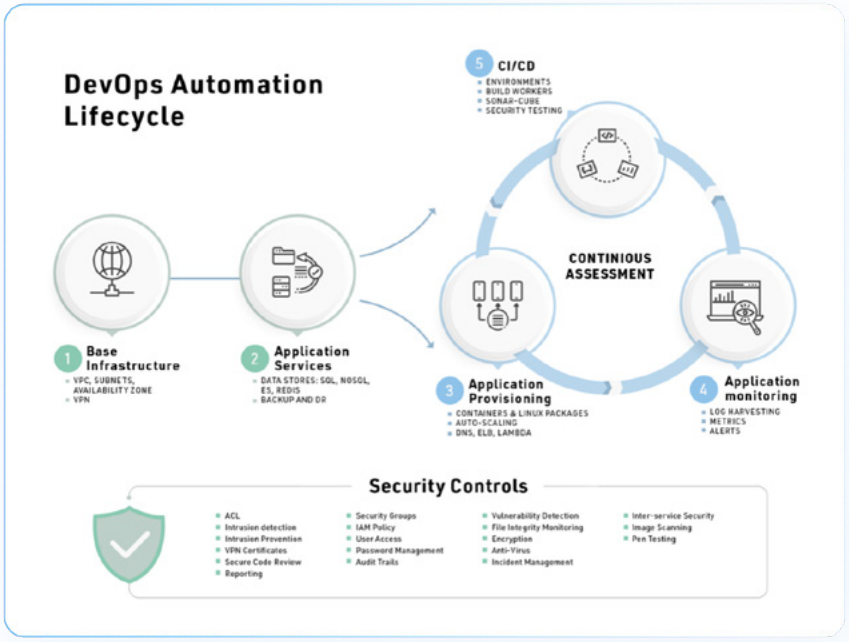


Figure 3

Base Infrastructure	Select the regions, spin up VPC/VNETs with proper address spaces, establish VPN connectivity, and set up availability zones.
Application Services	Define virtual machines, databases, NoSQL, object store, CDN, Elasticsearch, Redis, Memcached, message queues, and other supporting services. Public clouds have directed 90% of their investments in this area and an increasing number of clients are writing applications using these services for a faster Go-To-Market rate, robust scale, and better reliability. Other items in this area include disaster recovery, backup, image templates, resource management, and other supporting functions.
Application Provisioning:	Different automation techniques and tools can be applied depending on the application packaging type. For example: a. Kubernetes, EKS, and Azure Webapp for containerized workloads b. Amazon Lambda, Azure functions, and Google Cloud Functions for serverless workloads c. Databricks, EMR, Glue, etc. for Big Data use cases d. SageMaker, Kubeflow, and Google AI for AI use cases

Logging, Monitoring, and Alerts:	Core diagnostic functions that can be set up using several supported third-party tools. Centralized logging can be achieved by ELK, Sumo Logic, Splunk, and Datadog. We have Datadog, CloudWatch, SignalFx, and so on for monitoring and APM. For alerts we support Sentry. Many unified monitoring tools like Datadog supplies all these functions.
CI/CD	There are at least 25 good CI/CD tools in the industry from Jenkins to CircleCI, Harness.io, Azure DevOps, and so on. In this layer one also needs to put in place security testing pipelines that enforce secure coding practices via static code analysis and penetration testing.

SecOps Lifecycle and Compliance Frameworks

Organizations in non-regulated industries follow a set of best practices prescribed by in-house DevOps engineers. These are mandatory requirements and require much cost in DevOps labor to set up and support.

Some of the components that need to be created and monitored include security groups, IAM/AD policies, encryption, and some basic user access controls.

Regulated industries have published prescriptive frameworks for Cloud data security. They are exhaustive to implement and interpreting them in the context of a certain Cloud deployment requires deep subject matter expertise, often beyond the scope of the DevOps engineer.

Key Pain Point:	It is possible for just a single control, out of a requirement of hundreds, to be able to consume over 180 business days to implement.

Requirement 1: Install and maintain a firewall configuration to protect cardholder data	Milestone
1.1 Establish and implement firewall and router configuration standards that include the following:	
1.1.1 A formal process for approving and testing all network connections and changes to the firewall and router configurations	6
1.1.2 Current network diagram that identifies all connections between the cardholder data environment and other networks, including any wireless networks	1
1.13 Current diagram that shows all cardholder data flows across systems and networks	1
1.1.4 Requirements for firewall at each internet connection and between any demilitarized zone (DMZ) and the internal network zone	2
1.1.5 Description of groups, roles, and resposibilities for managment of network components	6
1.1.6 Documentation of business jsutification and approval for use of all services, protocols, and ports allowed. Including documentation of security features implemented for those protocols considered to be insecure.	2
1.1.7 Requirement to review firewall and router rule sets at least every six months	6

FIGURE 4: PCI Controls on AWS

Like PCI, different standards have their own control sets as shown in Figure 5.

Compliance Standard	Number of Controls	Stage for Control Implementation
PCI	300+	70% Provisioning time
HiTrust	150+	
HIPAA	50+	30% Post Provisioning time
NIST	900+	

Figure 5

70% of compliance controls must be applied during services and VM provisioning. This is a key element of compliance that catches many developers off-guard!

If these controls are missed at provisioning time, then typically reprovisioning is required. For example, disk encryption, placement of VMs in the right subnets, etc.

The DevOps function is approximately 70% security related. Unfortunately, virtually no standard security software like Prisma Cloud, Threat Stack, Laceworks, et al. has any role in provisioning!

Current State of DevOps and Infrastructure-as-code

The role of a DevOps engineer can be summarized in one sentence, per Figure 6.

> A DevOps engineer builds cloud infrastructure by stitching together a multitude of tools using his/her interpretation of best practices and standards.

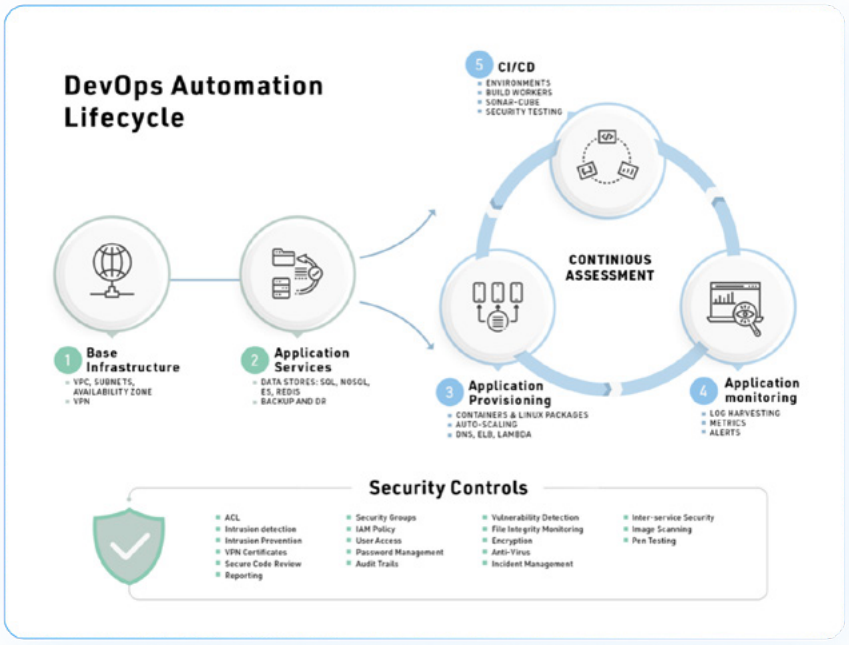


Figure 6: Examples of tools that are stitched together by DevOps

From 2010 to 2015, the most common approach to infrastructure automation was the use of templates wherein the operator gets a description of a desired configuration and inputs them in the form of templates. The key assumption is that the topology will not change and when it changes, those changes must be re-implemented out-of-band. Templates are acceptable for one-time setups, but people soon realized that infrastructure is constantly changing. Fragmented applications, microservices, and a plethora of cloud services add further volatility, leading to Infrastructure- as-Code (IAC).

To accommodate ever-changing infrastructure specifications, it was found that DevOps teams should treat the entire configuration as if one were building a software product. Engineering teams supply high-level specifications to DevOps. DevOps translates these specs into lower-level nuances where each detail is transcribed as code and follows a typical Software Development Life Cycle (SDLC), including code review, testing, and rollout, as shown in Figure 7.

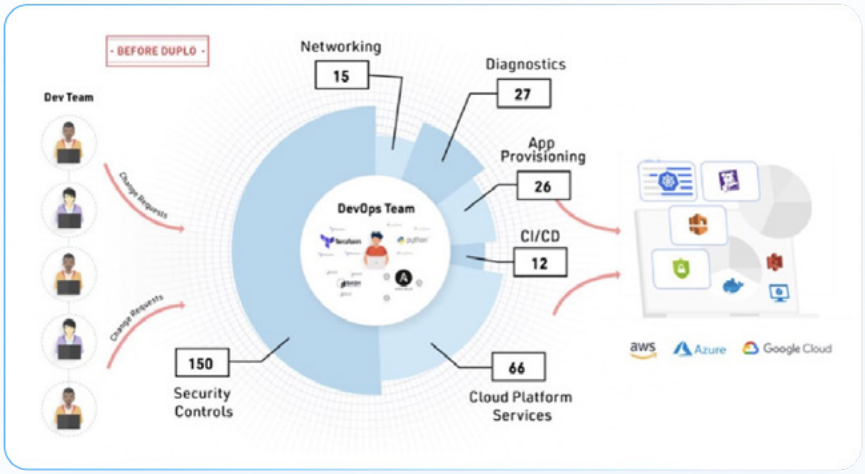


Figure 7

This approach has advantages:

- A single source, saved in a Git repository
- Declarative state
- Change tracking
- Repeatability

It also introduces a substantial set of disadvantages:

Increased subject matter expertise: For the operator, who now needs a programmer.

Open-ended and requires the operator to supply the lowest level of details: IAC is a programming language, and the onus is on the user to write the correct code. For example, one can create a security group open to the internet and IAC will not complain.

Longer change cycles: Many activities in operations are just-in-time and must be done by junior, lesser-skilled operators. For example, adding an IP to a WAF to prevent an attack, applying a patch to a server, or executing a script. If the user must update IAC, get a code review done, or do regression testing and rollout, then most operations teams fall short as they have neither the skill set nor the speed to address the needs of the hour.

Centralized control (Anti-pattern to Microservices): Pre-IAC pieces of infrastructure were configured and updated independently by different people in different shifts of operations. WAF, VMs, Containers, IAM, Security groups, databases, etc. are all different functions. Unfortunately, with IAC, even though it supports concepts like modules, the code base written by even the best DevOps engineers is like a monolith with a huge surface area. The scope of most state files is quite wide. Terraform, while promising, is still relatively in its infancy. There are no objects, classes, inheritance, or threads. Constructs as basic as loops and user-defined functions are hard to write. Figure 8 below compares this evolution to mainstream programming. In Terraform, everything is one block of monolithic code. One wrong "terraform apply" command can be catastrophic.

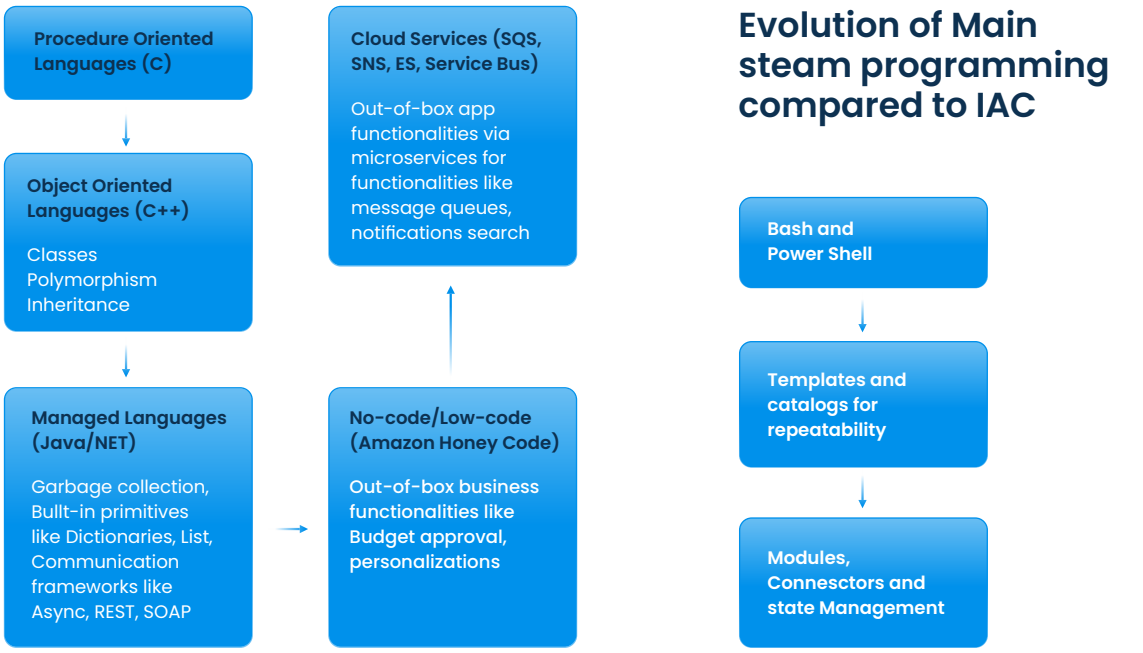


Figure 8

> Finding an engineer who is good at both programming and operations is like finding a unicorn. How often does one come across someone talking about objects, classes, and functions together with CIS benchmarks, IAM policies, and WAF? It is not surprising that there are upwards of 60,000 DevOps openings on LinkedIn.

DuploCloud: No-Code/Low-Code DevOps Automation Platform

At DuploCloud, we set out to address these problems and make IAC better. We envisioned a solution that has the following key elements:

Rules-based Engine: Translates a high-level application specification to low-level infrastructure constructs automatically, based on:

a. The cloud provider where the application is being deployed. The engine has well-architected framework rules for each supported cloud (AWS, Azure, GCP)
b. The application architecture at a level of abstraction shown in Figures 1 and 2
c. The desired compliance standard, such as PCI, HIPAA, GDPR, etc. in cloud providers. For example, the AWS PCI guide: https://aws.amazon.com/quickstart/architecture/compliance-pci/

State Machine: In cloud infrastructure today, almost nothing is done once. A state machine is essential for ongoing changes, detecting drifts, and remediation.

Application-Centric Policy Model: Compartmentalizes infrastructure constructs based on application boundaries. Figure 9 is the high-level DuploCloud policy model.

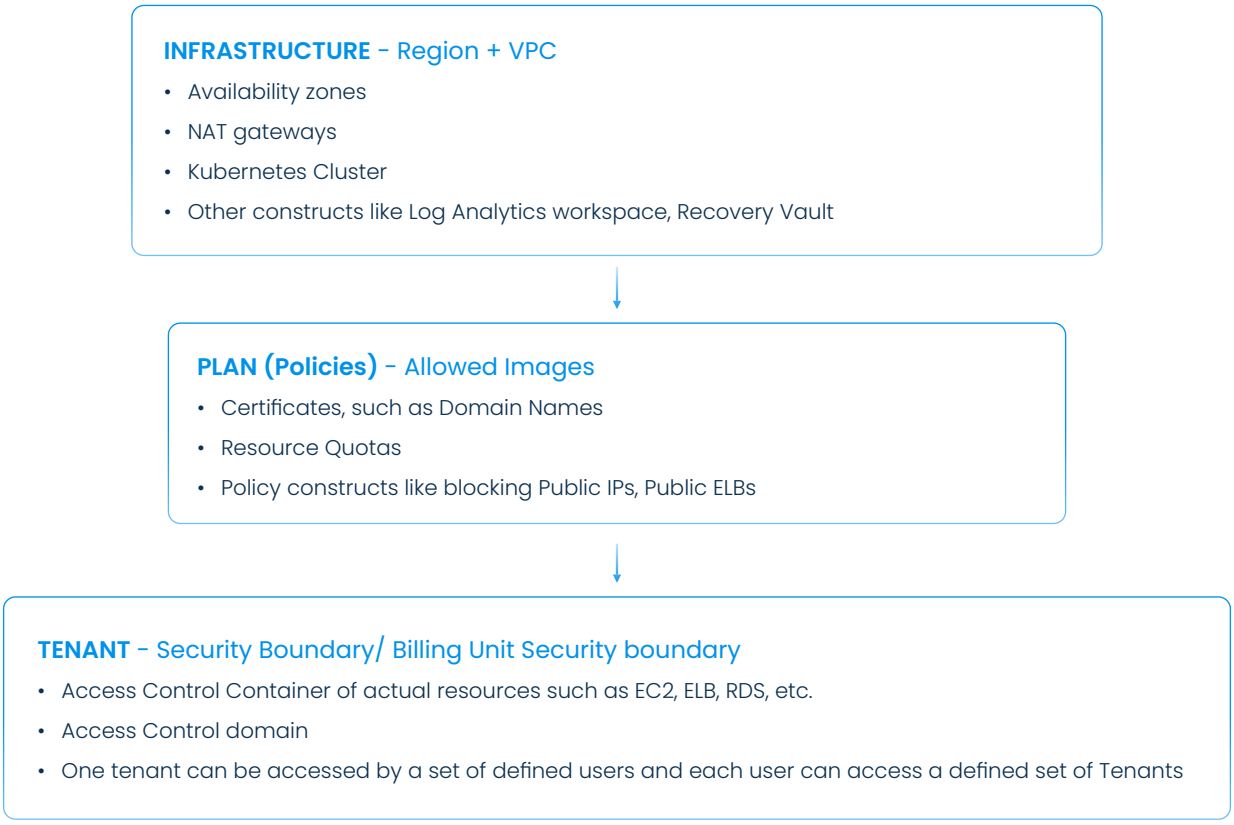


Figure 9

No Code UI: For users who do not want to manually write IAC, they can weave the E2E DevOps workflow using a web-based UI.

Low-Code IAC (Terraform provider/SDK): Using an SDK with built-in functions for best practices and compliance controls one can reduce the amount of Terraform code by over 90%.

An analogy of low-code DevOps: Terraform is like the C programming language where the user must do all memory management, and self-implement functions like HashMap, and dictionaries, while Java supplies an out-of-box implementation of these same constructs. For example, the user can instantiate a HashMap object by a single line of code. In the same way, DuploCloud supplies an SDK into Terraform where virtually all the best practices are built in.

For example, the code snippet in Figure 10 shows how one could create a new host via Terraform and request all host-level PCI controls with a single flag and request the host to be joined to an EKS cluster with a node selector. A complete example of building a topology using DuploCloud Terraform is described in the next section.

```
resource "duplocloud_aws_host" "host" "node01" {
        tenant_id                = duplocloud_tenant. tenant1. tenant_id
        friendly_name  = "eks-node01"
        image_id                 = data.duplocloud_native_host_image.ami.image_id
        capacity                 = "t3a.medium"
        agent_platform= eks
        keypair_type     = 2
        user_account   = duplocloud_tenant.tenant.tenant_id
}
```

Figure 10

Self-Hosted Solution: DuploCloud software deploys as a VM with admin privileges entirely within the customer's cloud account with no outside data management. Users interact with the software in one of the 3 ways:

a. Web Portal
b. Terraform using DuploCloud Provider
c. Rest API

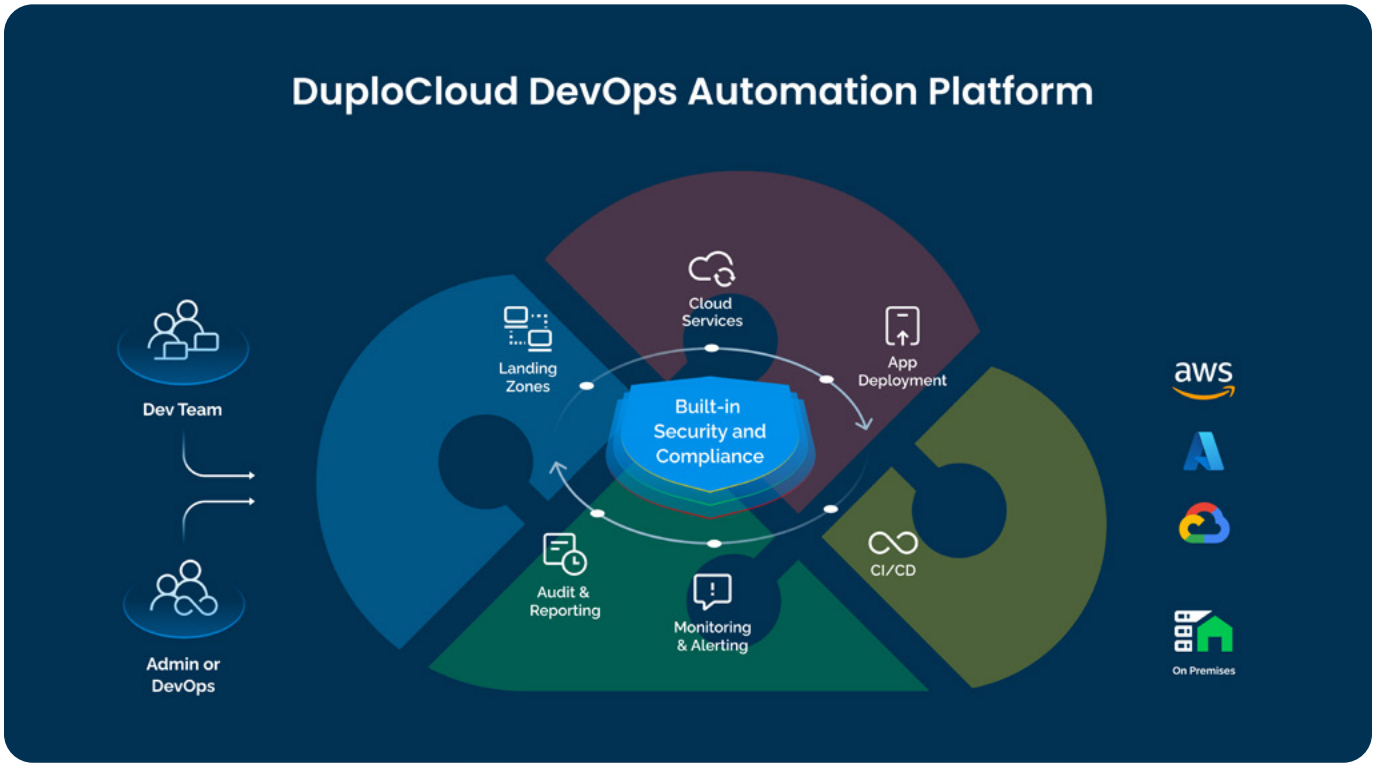


Figure 11

Demonstrating a Deployment with Low-Code and No-Code

Starting with an example topology in Figure 12 of an application in AWS, let's see how we can realize it first using no-code (DuploCloud Web Portal) and then using low-code (Terraform script with DuploCloud provider)

Deployment Topology: The application consists of a set of microservices to be deployed on EKS. The environment requires a VPC and 2 Availability Zones with 1 public and private subnet each. The database is hosted in AWS RDS, and S3 is the object store. All instances and containers are to be run in EC2 instances in private subnets and applications exposed to the internet via a load balancer that is fronted by a WAF. The environment needs PCI compliance according to the control set defined in the AWS PCI guide @ https://aws.amazon.com/quickstart/architecture/compliance-pci/. CloudWatch is to be used for metrics.

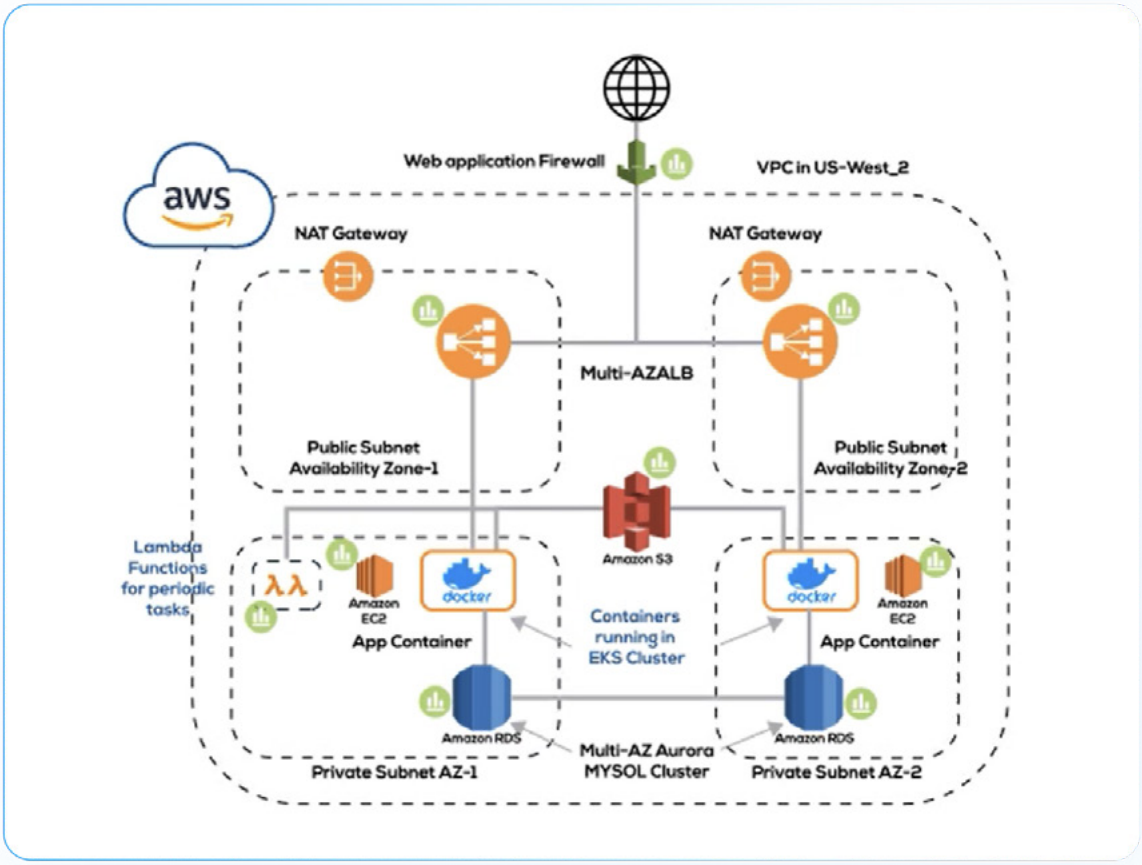


Figure 12

No-Code Deployment using Web Portal — Watch the following demo:

Low-Code Implementation: Figure 13 shows a code snippet that proves the same deployment can be achieved with about 100 lines of code which would have otherwise taken thousands of lines. For the sake of brevity, we only provide about 70% of the blueprint in this code snippet:

```
provider "duplocloud" {
        duplo_host              = "https://xxx.duplocloud.net"
        duplo_token             = "xxxx"}

resource "duplocloud_infrastructure" "finance" {
        infra_name              = "finance"
        cloud                   = 0
        region                  = "us-west-2"
        azcount                 = 2
        enable_k8_cluster       = true
        address_prefix  ="10.23.0.0/16"
        subnet_cidr             = 24
}
resource "duplocloud_teant" "invoice" {
        account_name = "invoice"
        plan_id         = "finance" }
```

Lines 5-13: Create an infrastructure named finance that includes a VPC in us-west-2 with 2 AZ with one public and one private subnet and an EKS cluster.

Lines 14-16: Create a tenant named **invoice** in the above infrastructure that will implicitly create security groups, IAM roles, instance profiles, KMS keys, Pem keys, a namespace in EKS, and many other placeholder constructs depending on the compliance framework to be followed.

```
18 resource "duplocloud_aws_host" "host1" {
19         tenant_id          = duplocloud_tenant.invoice.id
20         account_name = duplocloud_tenant.invoice.account_name
21         image_id           = var.eks_hardended_ami_id
22         capacity           = "t2.small"
23         friendly_name   = "host1"
24         agent_plataform    = eks
25         pci                = true
26 }
27 resource "duplocloud_duplo_service" "myservice" {
28         tenant_id          = duplocloud_tenant.invoice.id
29         name               = "myservice"
30         agent_plataform    = eks
31         docker_image   = "nginx:latest"
32         replicas           = var.my_service_replicas
33 }
```

Lines 18-26: Create a host in the invoice tenant and ask it to join the EKS cluster. The user specifies high-level parameters like name, capacity, and enable_pci, and internally the platform will apply the right set of security groups, IAM roles, instance profiles, user data to join to EKS, IAM policies, and a whole set of Host-based security software like vulnerability assessment, FIM, Intrusion detection and orchestrate the system to also collect these logs and register the node in a SIEM.

Lines 27-33: Create an EKS service or K8S Deployment using simple declarative user specifications. Behind the scenes the software will translate into EKS calls to deploy it in the right namespaces, set labels, and node selectors, affinities, etc.

Lines 34-49: Expose the service via an Application Load balancer where the user specifies what ports need to be exposed with health check URLs. Behind the scenes, the platform will auto-generate the nuances around node ports, Ingress, annotations, VPCs, subnets, security groups, and other details. Defaults for health check timeouts, health count, etc. were picked but could have been passed in the config.

Lines 51-56: Choose a DNS name for the service and attach it to one of the preexisting WAF IDs. Behind the scenes, Route53 programming of ALB Cnames, attachment of WAF, etc. is accomplished.

```
34  resource "duplocloud_duplo_service_lbconfigs" "serverlbs"  {
35              tenant_id                       = duplocloud_tenant.invoice.id
36              replication_controller_name             = duplocloud_duplo_service.myservice.name
37              lbconfigs  {
38                      certificate_arn = data.terraform_remote_state.tenant.outputs ["acm_certificate_arn" ]
39                      external_port       = 433
40                      health_check_urln = "/live"
41                      is_native = false
42                      lb_type = "alb"
43                      port = "3000"
44                      protocol = "http"
45                      is_internal = var.internal_lb
46                      service_name = duplocloud_duplo_service.server.name
47              }
48      # Workaround for AWS: Even after the ALB is available, there is some short duration where a V2 WAF cannot be
49  attached.   provisioner "local-exec"{command = "sleep10"}}
50
51  resource "duplocloud_duplo_service_params" "serverlbs"{
52              tenant_id                       = data.terraform_remote_state.tenant.outputs ["duplo_tenant_id"]
53              replication_controller_name = duplocloud_duplo_service_lbconfigs.serverlbs.replication_controller_name
54              dns_prfx = "api.${local.tenant_subdomain}"
55              webaclid = var.waf_v2_arn
56  }
```

Figure 13

Some one-time constructs, such as WAF, Cert in ACM, and Hardened AMI are done out-of-band and made available to the system to consume.

In these fifty-six lines of code in the above snippet, we have covered 70% of the topology shown in Figure 10. Setup and ongoing operations for this whole provisioning would still be less than three hundred lines of code. Without the DuploCloud provider, there would be thousands of lines of code.

Neither a PaaS nor a Restrictive Abstraction on the Cloud

A common drawback of many cloud management platforms is that while they supply the benefits of abstraction, they are quite restrictive if one must use cloud provider functions that have not been exposed by the platform. Most cloud management platforms are restricted to specific use case templates that must be created beforehand by an administrator and then exposed to the user. Anytime a change is needed in the topology the administrator must step in and update the templates.

DuploCloud is neither a PAAS nor does it come in the way of engineering teams and their cloud usage. Think of it like an SDK to Terraform or a DevOps (rules-based) engine that can autogenerate the lower-level DevSecOps nuances and boring compliance controls, while the engineering teams focus on building application logic using cloud-native services.

There are 3 reasons for DuploCloud's extreme flexibility:

Build and run arbitrary workflows	DuploCloud allows the combining of any of the services exposed by the cloud platform. No "pre-baked" templates need to be created by an administrator. For example, the blueprint shown in Figures 1 and 2 is arbitrary. Of course, workflows can be saved as templates and reused.
Incorporate new cloud services	DuploCloud's rules-based engine supports the addition of new cloud features effortlessly behind the scenes for most use cases. We simply add a set of JSON configurations that auto-generate code per the cloud provider configuration specifications and best practice guide for that feature set. This is the core of DuploCloud's IP. Further, these feature sets are based on cloud services and are not customer specific. For example, once Managed Kafka support is added in DuploCloud that is available to all customers.
Interlace Native Terraform with DuploCloud SDK	The advantage of having a self-hosted platform within one's cloud account means that the user is free to use the cloud resources in case a service is not exposed, or a workflow has some custom quirks. Within a single Terraform file, you can invoke the DuploCloud Terraform provider to supply a set of high-level resources and then add custom configurations using the cloud's Terraform provider. For example, in the code snippet shown in Figure 14 below, using the DuploCloud Terraform provider within a tenant, the user adds an EC2 instance attached to the EKS cluster, deploys a service, and exposes it via an ALB with a DNS name provisioned in Route 53. Then, in the existing ALB using the AWS Terraform provider, the user has added a new listener which redirects the URL https://abcorp.com to https://app.abcorp.com, which is provisioned as a CloudFront resource separately.

```
provider "aws" {
        version    = "~> 2.70"
        region     = var.region
}

provider "local" {
        version    = "~> 1.2"
}

provider "null" {
        version    = "~> 2.1"
}
....
....
....

resource "duplocloud_duplo_service_lbconfigs" "serverlbs"  {
        tenant_id                       = duplocloud_tenant.invoice.id
        replication_controller_name             = duplocloud_duplo_service.myservice.name
        lbconfigs  {
                certificate_arn = data.terraform_remote_state.tenant.outputs ["acm_certificate_arn" ]
                external_port       = 433
                health_check_urln = "/live"
                is_native = false
                lb_type = "alb"
                port = "3000"
                protocol = "http"
                is_internal = var.internal_lb
                service_name = duplocloud_duplo_service.server.name
        }
    # Workaround for AWS: Even after the ALB is available, there is some short duration where a V2 WAF
cannot be attached.   provisioner "local-exec"{ command = "sleep10" } }

     resource "aws_lb_listener_rule" "redirect-to-cloudfront" {

     listener_arn = data.aws_lb_listener.serverlbs[0].arn

     condition {
          host_header {
                values = [ local.tenant_fqdn ]
           }
     }

    action {
          type = "redirect"

          redirect  {
                protocol = "HTTPS"
                host = local.my_ui_fqdn
                status_code = "HTTP_301"
          }
      }
}
```

Figure 14

No Platform Lock-in

A common concern that customers have when using powerful technology is whether they are getting locked into a proprietary platform. Fortunately, we have addressed this concern with the ability to export native Terraform code with the state (state file) of the current infrastructure. This would be the scenario when the customer wants to wean off the DuploCloud solution for some reason and have their native Terraform with no proprietary constructs.

With the ability to export native terraform (IAC), disengaging with the DuploCloud platform is like disengaging a DevOps engineer, but still having access to the IAC code they wrote. Except that one would now have to hire a new engineer to support and evolve the automation with all the best practices, efficiency, and scale that is desired. The existing workloads are not affected.

A Comprehensive Platform that Continues to Evolve

DuploCloud supports virtually all common services in AWS, Azure, and GCP, with two to three services added each month. Any cloud service requested by our clients is added within two weeks.

Any configuration to be made in the cloud provider (AWS, Azure, or GCP), Kubernetes, or in a third-party tool supported natively (like OSSEC, Wazuh, ELK, ClamAV, Datadog, etc.) is within the scope of the platform. Historically 90% of the use cases have been served out-of-box in the software. For the rest there are 2 options (a) DuploCloud team will train and update the software with a typical turnaround time of 4-5 days. (b) The required configuration can be done directly on the cloud provider, K8S cluster, or the respective tool. An example of the 10% use case was when Managed Kafka was newly released, when a customer requested it to be exposed via DuploCloud, it took us 4-5 days.

Figure 15 shows the representative services around which customers have built workflows.

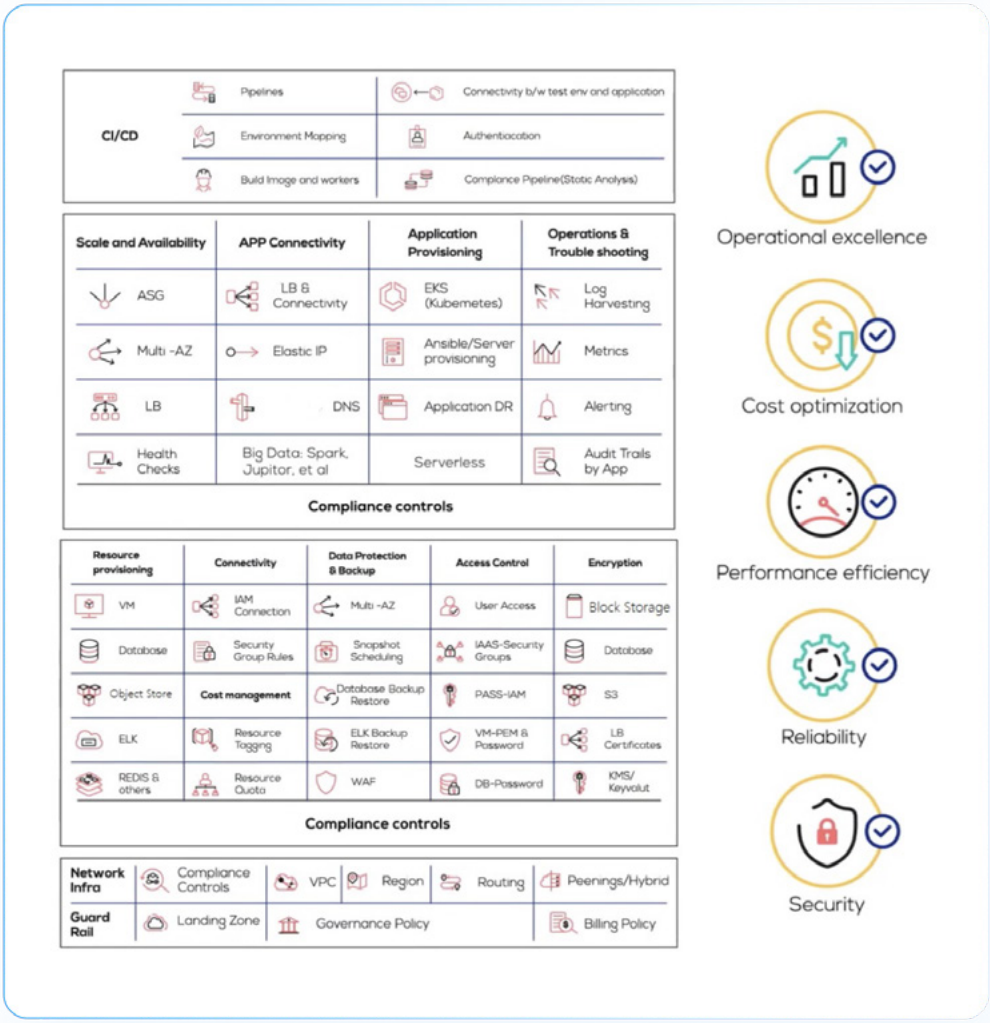


Figure 15

Summary

Every company is going through a digital transformation with a focus on moving to public clouds and achieving faster application delivery. With the growing demand for DevOps expertise, many enterprises are struggling to fill all their open positions needed to achieve the desired business goals. This skills shortage is slowing down overall application modernization, cloud migration, and automation projects which are critical for both business growth and to remain competitive.

DuploCloud supplies a new no-code-based approach to DevOps automation.

With over a hundred customers in regulated industries across Publicly listed enterprises, SMBs, and MSPs, we can show enormous productivity improvements across the board. Our customers can focus more on other application-related improvements instead of worrying about infrastructure, security, and compliance.

The three key advantages of using DuploCloud are:

- **10X faster infrastructure provisioning via automation**
- **Out-of-box secure and compliant application deployment**
- **70% reduction in cloud operating costs**

How About This

DuploCloud is the industry's only end-to-end low-code/no-code DevOps automation and compliance platform, designed to make DevOps and Infrastructure-as-Code accessible for everyone.

Founded in 2017 and built by the original engineers of Microsoft Azure and AWS, the software platform helps startups, SMBs, and companies that are building enterprise-grade applications or migrating to the cloud, save time and money.

The DuploCloud platform translates high-level application specifications into detailed and fully managed cloud configurations utilizing best practices around security, availability, and compliance guidelines.

Want to see DuploCloud in action?

Contact us at **info@duplocloud.net** or visit **duplocloud.com** to request a demo.