Provision Cloud Applications 10x Faster with a No-Code / Low-Code DevOps Platform

Contact us:
sales@duplocloud.net
duplocloud.com
# Table of Contents

**Introduction** .................................................................................................................................................................................. 3  
**The Trend Towards Modern Cloud-Based Deployments** .................................................................................................................. 4  
**Step One: Create an Application Blueprint** ................................................................................................................................. 5  
**DevOps Lifecycle** .............................................................................................................................................................................. 7  
**SecOps Lifecycle and Compliance Frameworks** ............................................................................................................................... 9  
**Current State of DevOps and Infrastructure-as-code** ......................................................................................................................... 11  
**DuploCloud: No-code/Low-code DevOps Software Platform** ............................................................................................................ 14  
**Demonstrating a Deployment with Low Code and No-Code** ............................................................................................................. 16  
**Neither a PaaS, nor a Restrictive Abstraction on the Cloud** ................................................................................................................ 20  
**No Platform Lock-in** ......................................................................................................................................................................... 22  
**A Comprehensive Platform that Continues to Evolve** .......................................................................................................................... 22  
**Summary** ....................................................................................................................................................................................... 24
Introduction

The myriad of today’s off-the-shelf components and services allows developers to create more complex applications that can work at scale either on-prem or in the cloud. While this provides great flexibility and agility in terms of application development, the proliferation of these components and services has created a drastic uptick in fragmentation throughout the infrastructure.

Applications that used to be built with just a few VMs for storage, compute and network have now exploded into scores of configurations around security groups, containers, namespaces, clusters, IAM roles, policies, object stores, NoSQL databases and so forth. In fact, building a fully automated and compliant infrastructure for as small as a 50 VM infrastructure in a regulated industry is a multi-month arduous process requiring uniquely skilled—and scarce—engineering resources.

Enter DuploCloud, a Low-Code/No-Code automation solution to DevOps that speeds cloud provisioning 10x, while lowering costs by 75%.

This whitepaper highlights the typical approach that companies take toward DevOps using infrastructure as code and the associated challenges. We then contrast that to DuploCloud’s approach to DevOps, an end-to-end platform that translates high level application specifications into detailed cloud configurations, incorporating best practices around security, availability, and compliance guidelines.

*The hyper-scale automation techniques described have been in use inside cloud providers like AWS and Azure for years, where just a thousand engineers are operating millions of workloads across the globe with top notch availability, scale, security, and compliance standards. The DuploCloud team were among the original inventors of these automation techniques in the public cloud and are now democratizing them for mainstream IT.*
The Trend Towards Modern Cloud-Based Deployments

Three major shifts are happening across all industries today:

1. Infrastructure is moving to the cloud and becoming 100% software driven (i.e., Infrastructure-as-Code)

2. Applications are getting more fragmented and diverse (aka, micro-services)

3. An ever-increasing number of application functionalities have moved out of the developer's code and into Platform Services, completely managed by cloud providers

With the increasing adoption of public clouds, we see enterprises wanting to move towards a state where cloud operations are 100% software driven. Applications are becoming a combination of micro-services using containers, managed services for databases, messaging, key-value stores, NoSQL stores, Lambda functions and object stores.
Step One: Create an Application Blueprint

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

Figure 1 shows a deployment architecture for an application in AWS. The topology consists of one set of microservices packaged as a Docker container running in ECS fargate. Aurora MySQL is used as a Database, services are exposed to the internet via an external load balancer which is fronted by a WAF. Another set of microservices could be using Lambda functions and API gateway. Data stores include S3 and DynamoDB in this case.
If an organization is in Azure, they have a deployment architecture that is Azure-specific. The constructs and terminology may change but conceptually it’s the same. One such topology is shown in Figure 2.

**FIGURE 2**

*Key Pain Point: The high-level architecture gets passed to DevOps teams which then translate these into 100s of lower-level cloud configurations that would require thousands of lines of Infrastructure-as-code. Deep subject matter expertise is required both in operations as well as 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 the application blueprints is typically done in a series of phases as shown in Figure 3.

1) **Base Infrastructure**: This is the starting point where one would pick the regions, spin up VPC/VNETs with right address spaces, VPN connectivity and setup availability zones.

2) **Application Services**: This is the area where we have our virtual machines, databases, NoSQL, object store, CDN, Elastic search, redis, memcache, 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 their applications using these for faster GTM, higher scale and reliability. Other items in this area include disaster recovery, backup, Image templates, resource management and other such supporting functions.
3) **Application Provisioning:** Depending on the application packaging type, different automation techniques and tools can be applied. For example:

   a) Kubernetes, Amazon ECS and Azure Webapp for containerized workloads
   
   b) Amazon Lambda and Azure functions for serverless workloads
   
   c) Databricks, EMR, Glue, etc. for big data use cases
   
   d) Sagemaker and Kubeflow for AI use cases.

4) **Logging, Monitoring and Alerts:** These are the core diagnostic functions that need to be set up. Centralized logging can be achieved by ELK, SumoLogic, Elastic search, Splunk and Datadog. For monitoring and APM we have Datadog, CloudWatch, SignalFX and so on. For alerts we have sentry. Many unified tools like Datadog provide all 3 functions.

5) **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 tend to follow a set of best practices determined by in-house DevOps engineers. These would be table stakes and include security groups, IAM/AD policies, encryption and some basic user access controls. Regulated industries have published a prescriptive framework for security in the cloud. While they are prescriptive, they are exhaustive to implement and interpreting them in the context of a certain cloud deployment requires deep subject matter expertise.

A screenshot of some of the controls is shown in Figure 4.

![FIGURE 4: PCI Controls on AWS](image-url)
Like PCI, different standards have their own control sets as shown in Figure 5.

<table>
<thead>
<tr>
<th>Compliance Standard</th>
<th>Number of Controls</th>
<th>Stage for Control Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCI</td>
<td>79</td>
<td>70% Provisioning time</td>
</tr>
<tr>
<td>HiTrust</td>
<td>80+</td>
<td></td>
</tr>
<tr>
<td>HIPAA</td>
<td>50+</td>
<td>30% Post Provisioning time</td>
</tr>
<tr>
<td>NIST</td>
<td>200+</td>
<td></td>
</tr>
</tbody>
</table>

**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, 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 whatsoever!
Current State of DevOps and Infrastructure-as-code

If one were to describe the role of a DevOps engineer 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 have to be re-implemented out-of-band. Templates are great for one-time setup, but people soon realized that infrastructure is constantly changing. Fragmented applications, microservices and plethora of cloud services added further volatility, leading to Infrastructure-as-Code (IAC).

To accommodate ever changing infrastructure specifications, it was determined that DevOps teams should treat the entire configuration as if one were building a software product. Engineering teams provide high level specifications to DevOps who then translate into lower-level nuances where each and every detail is written down as code and any change follows a
typical Software Development Life Cycle (SDLC) that includes code review, testing and rollout, as shown in Figure 7.

![Diagram](image)

**FIGURE 7**

This approach has some clear advantages:

- Single source of truth is saved in a git repository
- Declarative state
- Change tracking
- Repeatability

It also introduces a substantial set of disadvantages. Key among them:

- **Increased subject matter expertise:** for the operator role which now requires a programmer.

- **Open ended and requires the operator to provide the lowest level of details:** IAC is basically 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 are required to 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 go and update IAC, get a code review done, do some regression testing and rollout, then most operations teams will fall short as they will neither have the skill set nor the speed to address the need 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 even by the best DevOps engineers operates like a monolith with a huge surface area. The scope of most state files is quite wide. And 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 big block of monolithic code. One wrong “terraform apply” can be catastrophic.

![Evolution of Main stream programming compared to IAC](image)

**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 opening in LinkedIn alone.*
DuploCloud: No-code/Low-Code DevOps Software Platform

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

1. **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. as prescribed by cloud providers. For example, AWS PCI guide: [https://aws.amazon.com/quickstart/architecture/compliance-pci/](https://aws.amazon.com/quickstart/architecture/compliance-pci/)

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

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

![FIGURE 9](image-url)
4. **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.

5. **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, self-implement functions like HashMap, dictionaries, whereas Java provides 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 provides 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 ask all host level PCI controls with a single flag; ask the host to be joined to a 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" "host1" {
  tenant_id = duplocloud_tenant.tenant1.tenant_id
  user_account = duplocloud_tenant.tenant1.account_name
  image_id = "ami-062e7f29a4d477f5a"
  capacity = "t2.small"
  agent_platform = eks
  labels = "app01"
  friendly_name = "host1"
  pci = true
}
```

**FIGURE 10**

6. **Self-Hosted Solution:** DuploCloud software deploys as a VM with admin privileges entirely within the customer’s cloud account with absolutely no outside management of the data. Users interact with the software in one of the 3 ways:
   a. Web Portal  
   b. Terraform using DuploCloud Provider  
   c. Rest API
Figure 11 shows a graphical representation of the DuploCloud solution.

**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, 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 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 in the following demo:

https://vimeo.com/577816574
**Low-Code Implementation:** Figure 13 shows a code snippet which demonstrates 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 demonstrate about 70% of the blueprint in this code snippet:

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

resource "duplocloud_infrastructure" "finance" {  
infra_name = "finance"  
cloud = 0  
region = "us-west-2"  
account = 2  
enable_k8s_cluster = true  
address_prefix = "10.23.0.0/16"  
subnet_cidr = "24"  
}

resource "duplocloud_tenant" "invoice" {  
account_name = "Invoice"  
plan_id = "finance"  
}

resource "duplocloud_aws_host" "host1" {  
tenant_id = duplocloud_tenant.invoice.id  
account_name = duplocloud_tenant.invoice_account_name  
image_id = var.eks_harden_and_aml_id  
capacity = "t2.small"  
friendly_name = "host1"  
agent_platform = "eks"  
pci = true  
}

resource "duplocloud_duplo_service" "mYServicE" {  
tenant_id = duploCloud_tenant.invoice.id  
name = "my-service"  
agent_platform = "eks"  
docker_image = "nginx:latest"  
replicas = var.my_service_replicas  
}

resource "duplocloud_duplo_service_lconfigs" "server1ns" {  
tenant_id = duplocloud_tenant.invoice.id  
replication_controller_name = duplocloud_duplo_service.myservice.name  
}

lconfigs {  
  certificate_arm = data.terraform_remote_state.tenant.outputs["my_certificate_arm"]  
  external_port = 80  
  health_check_url = "/live"  
  is_alive = false  
  lb_type = "alb"  
  port = 8080  
  protocol = "https"  
  is_internal = var.internal_lb  
  service_name = duplocloud_duplo_service.server.name  
}

# Workaround for ALB: Even after the ALB is available, there is some short duration when a V2 WAF cannot be attached.  
provisioner "local-exec" {  
  command = "sleep 10"  
}

resource "duplocloud_duplo_service_params" "server1ns" {  
tenant_id = data.terraform_remote_state.tenant.outputs["duplo_tenanT_id"]  
replication_controller_name = duplocloud_duplo_service_lconfigs.serverlbs.replication_controller_name  
dns_prefix = var.my.service_subdomain  
webaclid = var.waf_v2_arn  
}
```

**FIGURE 13**

- **Lines 5-13:** Create “infrastructure” called “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” called “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.

- **Lines 18-26:** Create a host in the “invoice” tenant and ask it to be joined to the EKS cluster. User specifies high level specific parameters like name, capacity, 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 label 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.

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 56 lines of code in the above snippet we have covered 70% of the topology shown in Figure10. Setup and ongoing operations for this whole provisioning would still be less than 300 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 provide the benefits of abstraction, they are quite restrictive if one must leverage cloud provider functions that have not been exposed by the platform. In fact, 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 auto-generate 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:

1. **Build and operate 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.

2. **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 in fact 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.

3. **Interlace Native Terraform with DuploCloud SDK.** The advantage of having a self-hosted platform within one’s own 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, one can invoke the DuploCloud Terraform provider to provision 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 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](https://abcorp.com) to [https://app.abcorp.com](https://app.abcorp.com) which is provisioned as a CloudFront resource separately.

```hcl
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["scm_certificate_arn"]
    external_port = 443
    health_check_url = "/live"
    is_native = false
    lb_type = "Alb"
    port = "3000"
    protocol = "http"
    is_internal = var.internal_lb
    service_name = duplocloud_duplo_service.myservice.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 = "sleep 10s" }
}

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_al_fqdn
      status_code = "HTTP_301"
    }
  }
}
```

**FIGURE 14**
No Platform Lock-in

A common concern that customers have when using a 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 terminating 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 maintain and evolve the automation with all the best practices, efficiency and scale that is desired. The existing workloads are not impacted.*

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 Duplo it took us 4-5 days.*
Figure 15 shows the representative services around which customers have built workflows.

<table>
<thead>
<tr>
<th>CI/CD</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Pipelines</td>
<td>Connectivity b/w test env and application</td>
</tr>
<tr>
<td>Environment Mapping</td>
<td>Authentication</td>
</tr>
<tr>
<td>Build Image and workers</td>
<td>Compliance Pipeline (Static Analysis)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Scale and Availability</th>
<th>APP Connectivity</th>
<th>Application Provisioning</th>
<th>Operations &amp; Trouble shooting</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASG</td>
<td>LB &amp; Connectivity</td>
<td>EKS (Kubernetes)</td>
<td>Log Harvesting</td>
</tr>
<tr>
<td>Multi-AZ</td>
<td>Elastic IP</td>
<td>Ansible/Server provisioning</td>
<td>Metrics</td>
</tr>
<tr>
<td>LB</td>
<td>DNS</td>
<td>Application DR</td>
<td>Alerting</td>
</tr>
</tbody>
</table>

Compliance controls

<table>
<thead>
<tr>
<th>Resource provisioning</th>
<th>Connectivity</th>
<th>Data Protection &amp; Backup</th>
<th>Access Control</th>
<th>Encryption</th>
</tr>
</thead>
<tbody>
<tr>
<td>VM</td>
<td>IAM Connection</td>
<td>Multi-AZ</td>
<td>User Access</td>
<td>Block Storage</td>
</tr>
<tr>
<td>Database</td>
<td>Security Group Rules</td>
<td>Snapshot Scheduling</td>
<td>IAM Security Groups</td>
<td>Database</td>
</tr>
<tr>
<td>Object Store</td>
<td>Cost management</td>
<td>Database Backup Restore</td>
<td>PASS-IAM</td>
<td>S3</td>
</tr>
<tr>
<td>ELK</td>
<td>Resource Tagging</td>
<td>ELK Backup Restore</td>
<td>VM-FLM &amp; Password</td>
<td>LB Certificates</td>
</tr>
<tr>
<td>Redis &amp; others</td>
<td>Resource Quota</td>
<td>WAF</td>
<td>DB-Password</td>
<td>KMS/KeyVault</td>
</tr>
</tbody>
</table>

Compliance controls

<table>
<thead>
<tr>
<th>Network Infra</th>
<th>Guard Rail</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compliance Controls</td>
<td>VPC, Region, Routing, Peering/Hybrid</td>
</tr>
<tr>
<td>Landing Zone</td>
<td>Governance Policy, Billing Policy</td>
</tr>
</tbody>
</table>

**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 provides a new no-code based approach to DevOps automation.

With several dozen customers in regulated industries across Publicly listed enterprises, SMBs and MSPs, we can show enormous productivity improvements across the board. Our customers can do much more with such that in-house DevOps teams can focus more on other application-related improvements instead of worrying about infrastructure, security, and compliance.

The three key advantages of using DuploCloud are:

1. 10X faster automation
2. Out-of-box secure and compliant application deployment
3. 70% reduction in cloud operating costs