# **DevOps -Cloud Design**

**Table of Contents**

[**DevOps -Cloud Design** 1](#_Toc65450089)

[**1** **Introduction** 3](#_Toc65450090)

[**2** **DevOps Road Map** 3](#_Toc65450091)

[**3** **CI/CD solution** 3](#_Toc65450092)

[**4** **Deployment Methodology** 4](#_Toc65450093)

[**5** **Branching Strategy** 4](#_Toc65450095)

[**6** **Build Process** 4](#_Toc65450096)

[**7** **Quality Assurance** 5](#_Toc65450097)

[**8** **Release Process** 5](#_Toc65450098)

[**9** **CI/CD Pipeline to Amazon Elastic Kubernetes Service** 6](#_Toc65450099)

[**9.1** **Amazon Fargate Elastic Kubernetes Service (EKS)** 6](#_Toc65450100)

[**9.2** **CI Flow** 7](#_Toc65450101)

[**9.3** **CD Flow to EKS** 7](#_Toc65450102)

[**9.4** **Fargate EKS** 7](#_Toc65450103)

[**9.5** **Pros and Cons of EKS** 9](#_Toc65450104)

[**9.6** **ECS Vs EKS Vs Fargate in a Nutshell** 9](#_Toc65450105)

[**10** **Cloud Watch Monitoring** 10](#_Toc65450106)

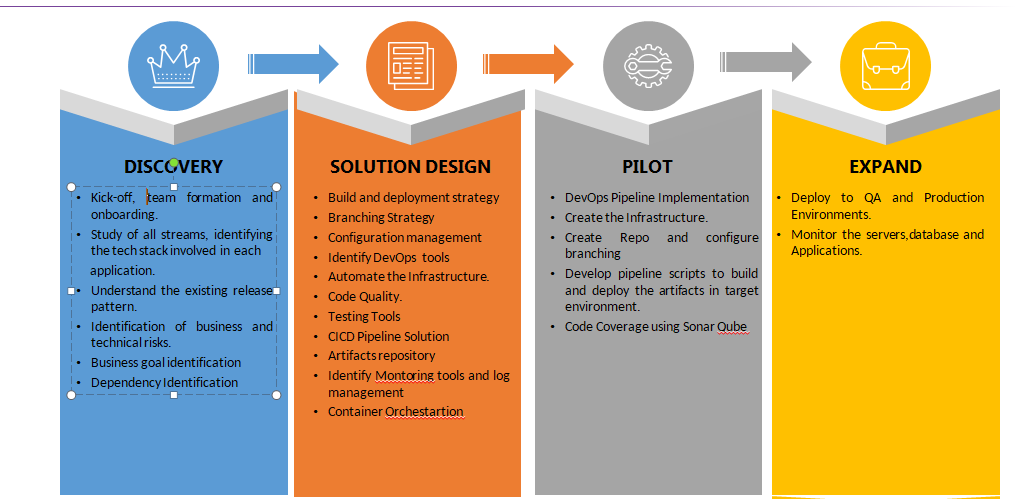
# **Introduction**

The purpose of this document is to describe the high-level solution for implementing continuous integration and continuous deployment AWS EKS cluster. This document covers our overall technical strategy for architecting the cloud infrastructure.

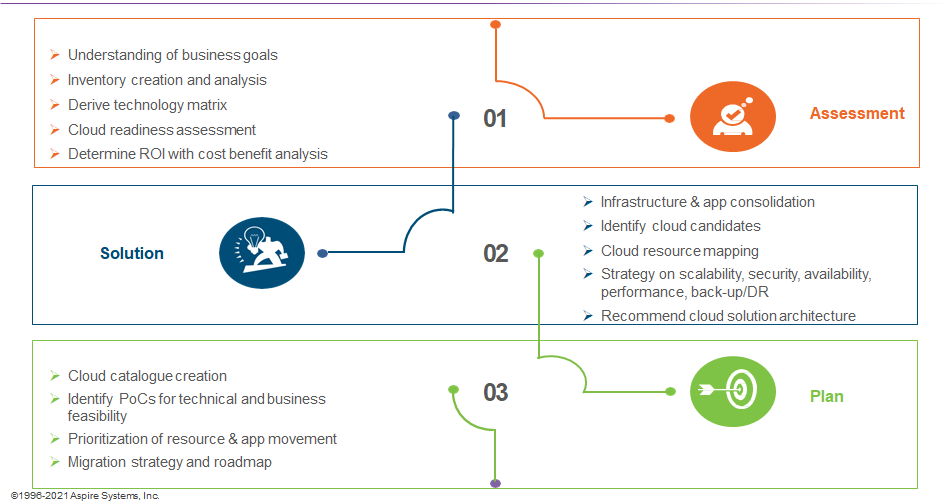
Aspire Systems transforms the existing Applications into DevOps model. Deliver a detailed roadmap and Implementation plan to

* Capture the current as-is state of release management processes and production operations.
* Improvements of Automated CICD
* Using EKS manage and orchestrate Kubernetes cluster on AWS Infrastructure
* Improvements in monitoring and logging solutions to enhance Cloud Watch Monitor.
* Experience in deploying and managing micro-services based applications.

# **DevOps Road Map**



# **Cloud Migration Roadmap**



# **CI/CD solution**

* The platform should help automate the release and delivery of applications, shortening the delivery lifecycle, streamlining manual processes, and accelerating team velocity.
* The platform will support source code management which enables coordination, sharing, and collaboration across the development teams. It should also track and merge branches, audit changes, and enable concurrent work, to accelerate software delivery.
* The platform should enable teams to package their applications and dependencies and build artifacts with ease to ensure DevOps acceleration with automated software pipelines.
* The platform should be able to automate unit testing and code coverage and integrate these into the pipeline.

# **Deployment Methodology**

# The components/products are deployed in various environments using a linear strategy. The same artifact is promoted between all stages when the previous deployment is successful. As per our understanding, components deployments traverse through Dev/QA/PRE PROD/PROD Environments

# **Branching Strategy**

The Master branch is a stable one that has the latest code of production environment. The develop branch is an intermediate one that gets created from the master. When the development is over, a release branch will be created from this branch. Hotfix branch is created out of master when patch release needs to be pushed.

# **Build Process**

The local development packages are compiled locally and the build artifacts are pushed to the centralized Jfrog Artifacts repository.

# **Quality Assurance**

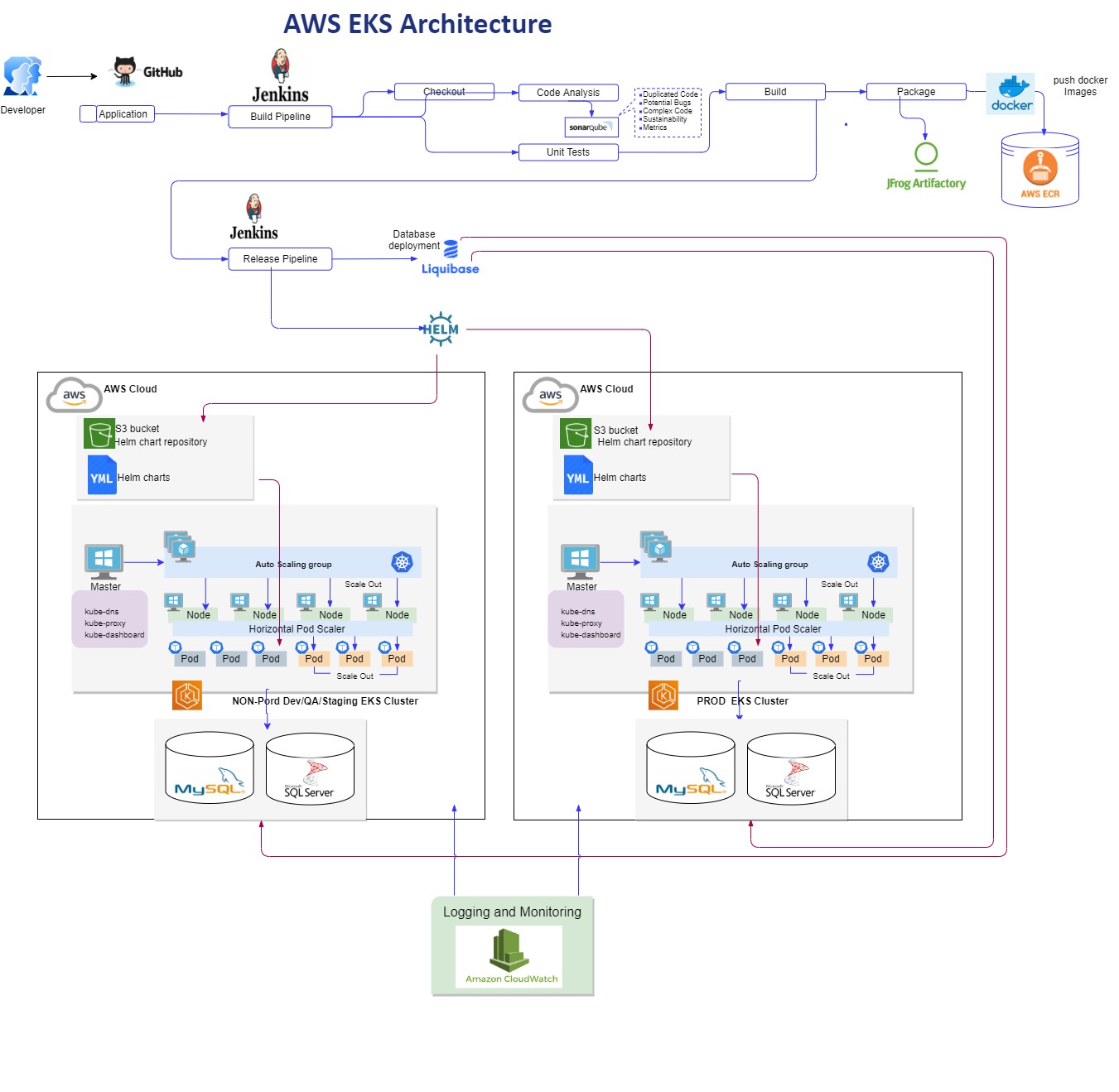
There are plenty of test cases are executed in each stage. The testing team starts writing the test cases in parallel with the development activity. Therefore, the test steps are aligned with the requirement most of the time, and test cases are approved internally before executing in any environment.

# **Release Process**

A developer creates a new branch in the components-specific repository. Ideally, the new branch will be created from master otherwise, the latest release branch. Once the new branch is created, the developer will clone the repository into their local development environment and start pushing custom changes. When the development is complete, the Pull Request will be created from the current working branch to the destination branch (e.g. master). Eventually, all the members who are working in CI/CD will be added as a reviewer and based on no conflicts the Pull request will be merged to the destination branch.

# **CI/CD Pipeline to Amazon Elastic Kubernetes Service**

## **Amazon Fargate Elastic Kubernetes Service (EKS)**



## **CI Flow**

The following shows the automation steps involved in the package deployments. All stages will be orchestrated from Amazon pipelines.

* Developers push the code to source code repository GitHub repository.
* Jenkins pipelines combine application and runs the unit test and validate all the quality code and start the CI pipeline build process.
* Enabled gated release with code Quality using SonarQube.
* The generated Artifacts are stored in Jfrog Artifacts.
* A docker image is created from the pipeline and it is pushed to an Elastic Container Registry.

## **CD Flow to EKS**

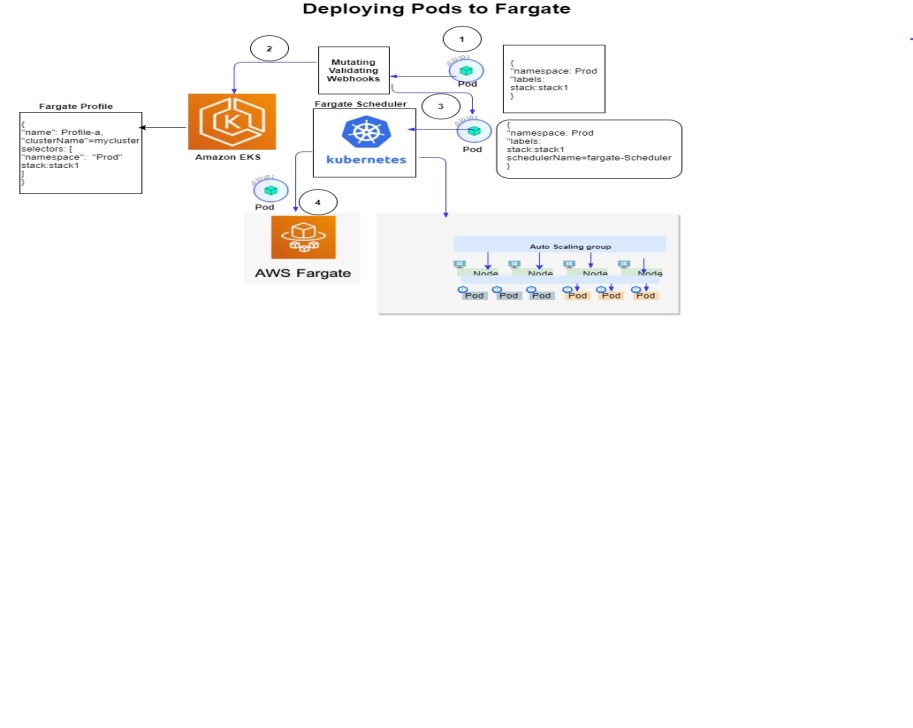
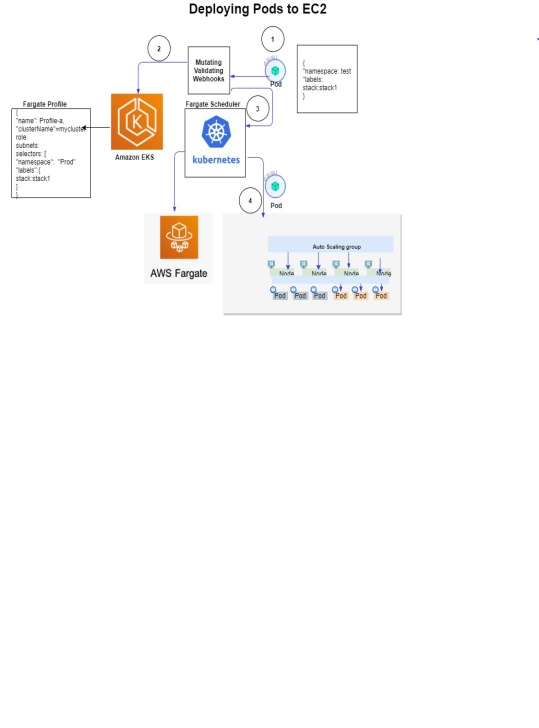
* Create EKS cluster which represents control plane
* Aws will Provision & manages Kubernetes Master Nodes replicated across multiple availability zones.
* EKS master nodes and control plane need to be configured.
* Worker Nodes and Master Nodes are get connected and deploy the containers in the cluster.
* Deploy the Application using deployment yml file which contains specification to the docker image which is stored in ECR and service yml file of type load balancer.
* Create Autoscaling for Pods and Cluster Autoscaler using yml files.
* Helm Charts define,Install and upgrade complex Kubernetes application.
* Create Helm charts chart.yml,values.yml and manifest templates.
* Store Helm charts in Amazon S3 bucket
* Helm charts are the deployment orchestrator for Kubernetes. The pod definition files will be in the form of helm charts and this will be updated now and then with the latest changeset.
* Update the helm chart with new image URL and update the helm chart version as well.
* Depoly the application in EKS Cluster and access your applications.
* Monitor an EKS cluster using Prometheus and Grafana.
* Liquibase is used to manage and deploy database changes on MySQL & SQL Server on AWS Cloud.

## **Fargate EKS**

Deploy the pods on Prod,Stage,Dev Environments based on the Fargate Profile.

The below diagram shows Pods deployed mixed on both EC2 and Fargate depends on Prod & Non Prod Environment.

* Deploying Pods to EC2
* Depolying Pods to Fargate



* Fargate EKS as a serverless offering, this platform will provide lower costs as you only pay for what you use. It also takes away the complexities of managing hosts and clusters including a Kubernetes cluster.
* A serverless container orchestration service offering is based on EKS technology and Kubernetes.  
  If you want to deploy Kubernetes pods without worrying about cluster management, then this may be a good choice since it provides a managed control plane and data plane so you can just focus on building containerised applications.
* AWS Fargate is best described as a serverless compute engine for containers.
* The service removes the need to provision and manage servers, and lets you specify and pay for resources per application.
* Its this feature that makes the service attractive to developers as it allows them to focus solely on application design — leaving the provisioning and management of servers to AWS.
* Fargate with EKS services and is the latest service allowing users to deploy Kubernetes pods onto Fargate without worrying about the management of clusters and servers.
* Fargate provides a managed control plane and data plane which means you will not see the EC2 cluster nodes it uses to deploy your pod to. Your pods are deployed to an AWS-owned account against a fleet of VMs and then gets connected to your VPC via an ENI (Elastic Network Interface).
* The advantage here is there are no nodes to manage. The unit of charge is the pod itself. When you deploy a pod there is a 1:1 mapping, with one pod deployed to a single micro VM.
* AWS Fargate EKS is serverless container data plane which runs your container on their managed infrastructure without the need for you to provision and maintain your own cluster.
* Amazon EKS is traditional container data plane with managed or unmanaged nodes. There is no visiblity of the EC2 of nodes when it comes to managing a contol plane. However there is more work with have to manage a Kubernetes cluster.
* With the introduction of Fargate EKS you now have ability to deploy to either AWS Fargate or Amazon EKS, or both. The key parameters to take note here are:
* Defining a Fargate profile This allows Fargate to understand the AWS infrastructure and assign a scheduler i.e. EKS or Fargate.
* Associated Namespace This ties back to the namespace defined within the pod.

## **Pros and Cons of EKS**

|  |  |
| --- | --- |
| Pros | Cons |
| AWS Managed Service. No management overhead | Cost for Managed Control Pane ($0.10 per hour for each Amazon EKS cluster) |
| Large open source community support | No out of box dashboard to monitor the cluster. Rely on third party monitoring system |
| K8s POD design always ensures that multiple containers (inside a POD) are scheduled on the same node, which reduces the latency between data processing. |  |
| Supports both EC2 and Fargate worker node model |  |
| Easy to migrate from one cloud to another or On-Prem, since the deployment files doesn’t need bigger changes |  |
| Autoscaler helps to scale the underlying EC2 nodes based on the application load |  |
| Ability to assign roles to PODS to establish fine-grained access control to AWS resources ( Eg: S3, Paramstore) |  |
| Horizontal and Vertical POD scaling to manage peak load |  |
| Most widely used and proven orchestrator |  |

## **ECS Vs EKS Vs Fargate in a Nutshell**

| Amazon ECS | Amazon EKS | Amazon FARGATE |
| --- | --- | --- |

|  |  |  |
| --- | --- | --- |
| The Good | | |
| Popularly Known as Amazon's Docker as a service. Few Call them, Amazon Beanstalk in multi-docker mode too. | Popularly known as Amazon's Kubernetes as a service. | Dev folks dearly call it the The Container Manager. |
| Offers Support in its CLI for Docker Compose. | Offers all the features of ECS, plus VPC for pod networking and isolation, at the cluster level. | Offers the same API actions as ECS, so you can use the ECS console or CLI, or the AWS CLI. |
| Supports duplicating environments using AWS CLI/SDK calls, thus helps in managing hundreds of containers | Supports upstream Kubernetes and replicates across three masters in different Availability Zones | Supports heterogeneous clusters that are made up of tasks running on both EC2 and Fargate launch types. Ideal for rapid horizontal scaling. |
| Integrates Seamlessly with ECR. This eases custom Docker images management. | Gives the advantage of running the same scheduler in AWS or anywhere else. | Helps to focus on designing and building your applications instead of managing the infrastructure that runs them. |
| Eliminates own registry management. | Can replicate container environment to another live environment in AWS, with minimal modification(s) | Takes care of bin packing problem. |
| Has auto healing feature, so failed containers will be relaunced automatically. | Extracts an additonal layer of scheduling and clustering to a container environment. | Supports AWS vpc network mode natively, which means all of the tasks running on the same instance share the instance's elastic network interface (ENI). |
| it's for free, but you have to pay for underlying resources provisioned to support applications. | Each cluster costs just $0.20 per hour. The major advantage over ECS is that a single Amazon EKS cluster is sufficient to run multiple applications. | Pay for the computing time, rather than the underlying EC2 instances. Works out cheaper, But can spiral out of control, depending on the usecase. |
|  | All communications between pods are via IP addresses in the VPC | Unlike ECS, Fargate has its own fleet of EC2s ready for your tasks. You can provision tens or thousands of containers in seconds. |
| The Bad | | |
| Not Easy to work with distributed systems. | Does not offer as deeper integration into the AWS compared to ECS. | Tasks must be launched into a cluster, even through it abstracts away VMs. |
| While scaling the service the service, you will have to wait until a new EC2 instance is deployed to launch a new task in that instance. | Charges applicable for launching complementary resources, like EBS volumes. | Pricing is based on the memory and CPU required to run a tasks, as well as the duration the task runs (by second and a minimum of 1 minute). If you launch complementary resources, like load balancers, you'll be charged for that as well. |
| Cannot relocate container instances to a different cluster. Neither can you change the instance type after launching. | Can spin up only three cluster in a region, currently. If need be, you can spin up more than three clusters but only after raising a ticket. | P.S AWS announced price reduction of up to 50% sometime during Jan 2019. Check for your use case keeping both CPU and memory utilization in mind. |
|  | Maximum number of control plane security groups per cluster is five. |  |
| The Ugly | | |
| Running own service discovery has ELB/ALB costs attached for services that doesn't need to be exposed outside. | Assigning pod-level IAM is a difficult task. | Customization options are less. |
| Getting on- demand clusters is time consuming. |  | Required to run your own components. |
|  |  | Has long startup times |
|  |  | No persistent filesystem access. |

# **Cloud Watch Monitoring**

* **Amazon Cloud Watch** allows you to access all your data from a single platform. It is natively integrated with more than 70+ AWS services. Amazon CloudWatch Logs let you monitor, process, store, and access log files from Amazon EC2 instances, AWS CloudTrail, Lambda functions, VPC flow logs, or other sources. With CloudWatch Logs, Cloud Administrator can troubleshoot AWS infrastructure and applications using an existing system, application, and custom log files from applications.
* **Metrics** You can view statistical graphs of published metrics with the AWS Management Console. CloudWatch stores data about a metric as a series of data points. Each data point has an associated time stamp. You can even publish an aggregated set of data points.
* **Alarms** CloudWatch Alarms feature allows to watch CloudWatch metrics and to receive notifications when the metrics fall outside of the levels (high or low thresholds) that is configured. Our AWS consultants can attach multiple Alarms to each metric and each one can have multiple actions
* **Auto Recovery** can be implemented with the help of CloudWatch. If an instance fails a system status check, you can use CloudWatch to automatically reboot or recover it.