AWS Whitepaper

Continuous Integration and Continuous Delivery for 5G Networks on AWS

Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Continuous Integration and Continuous Delivery for 5G Networks on AWS: AWS Whitepaper

Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

Continuous Integration and Continuous Delivery for 5G Networks on AWS

Publication date: **March 08, 2021** (*Document revisions*)

Abstract

This whitepaper introduces continuous integration and continuous delivery (CI/CD) for 5G networks, and how Amazon Web Services (AWS) tools and services can be used to fully automate the deployment and upgrades of 5G network functions. The whitepaper provides detailed description of the different stages of CI/CD for 5G network functions, including network setup, infrastructure deployment, cloud-native network functions deployment, and continuous updates of network functions. It also provides details on integration with open-source and third-party tools for testing, observability, and orchestration.

This whitepaper is aimed at Communication Services Providers (CSPs) as well as Independent Software Vendors (ISVs).

Introduction

Historically, development, lab and field integration testing, and production deployment of new network nodes or new features in a cellular network took weeks or even months to ensure the stability of mission and business critical telecommunications (telecom) services. The long cycle of deployment was caused by the monolithic architecture of traditional network nodes, a multi-vendor environment, and many point-to-point interfaces among network entities in the 2G, 3G, and 4G mobile networks.

As introduced in the *5G Network Evolution with AWS* whitepaper, 5G mobile networks, as standardized by 3GPP, now support a cloud-native architecture enabled by virtualization and containerization. More specifically, 5G networks introduce and support a new paradigm of the microservice, stateless, and service-based architecture.

This 5G architecture means that different network functions can work as loosely coupled independent services that communicate with each other through well-defined interfaces and APIs. Most importantly, each network function can be updated independently. This architecture shift in 5G enables CSPs to achieve more agility and operational efficiency by making it easier to roll out updates for network functions more frequently, while maintaining the testing, security requirements, and standards through automation.

Integration and deployment of new features for a CSP generally start when the network function vendor releases a new network function software package, such as a Docker image in a container-based network function, or a new configuration file, such as a Helm chart in the Kubernetes application case. (A Helm chart is a collection of files that describe a related set of Kubernetes resources).

The idea of using the paradigm of CI/CD for 5G network function deployment is gaining traction, but the practical realization of this idea has been a challenge in the telecom industry.

AAWSWS has pioneered the development of new CI/CD tools for software delivery to help a broad spectrum of industries develop and roll out software changes rapidly, while maintaining systems stability and security. These tools include a set of Software Development and Operations (DevOps) services such as AWS CodeStar, CodeCommit, CodePipeline, CodeBuild, and CodeDeploy.

AWS also evangelizes the idea of Infrastructure as Code (IaC) using the AWS Cloud Development Kit (AWS CDK), AWS CloudFormation, and API-based third-party tools such as Terraform. Using these tools, AWS can store the deployment processes of network function within AWS as a source code, and maintain this IaC source code in the CI/CD pipeline to realize continuous delivery.

This whitepaper describes detailed processes for leveraging AWS IaC and CI/CD tools for the deployment and update of the 5G network function. Additionally, this whitepaper covers integration with third-party tools for testing, observability, and orchestration.

AWS CI/CD tools are not restricted to 5G network functions. They are also employed for automating the deployment of 4G networks, which enables CSPs to rapidly and efficiently deploy and update 4G network functions. Most 4G network functions are Virtual Network Function (VNF) based. AWS CI/CD toolsets like AWS CloudFormation can be used to automate the deployment of 4G VNFs, bringing scale and time efficiency for 4G network deployments.

Continuous Integration and Continuous Delivery

Continuous Integration

Continuous Integration (CI) is a software process in which developers regularly push their code into a central repository such as AWS CodeCommit or GitHub. Every code push triggers an automated build, followed by the running of tests. The main goal of CI is to discover code issues at an early stage, improve code quality, and reduce the time it takes to validate and release new software updates.

Continuous Delivery and Deployment

Continuous Delivery (CD) is a software process in which artifacts are deployed to the test environment, staging environment, and production environment. Continuous delivery can be fully automated, or have approval stages at critical points. This ensures that all required approvals prior to deployment, such as release management approval, are in place. When continuous delivery is correctly implemented, developers always have a deployment-ready build artifact that has passed through a standardized test process.

With *Continuous Deployment*, revisions are deployed to a production environment automatically without explicit approval from a developer, making the entire software release process automated. This allows for a continuous customer feedback loop early in the product lifecycle.

With Continuous Deployment, every change that is committed and passes automated tests is released to production automatically. Continuous Delivery is not meant to release every change that is committed and pass automated tests to production immediately, but to ensure that every change is ready to go to production.

Infrastructure as Code

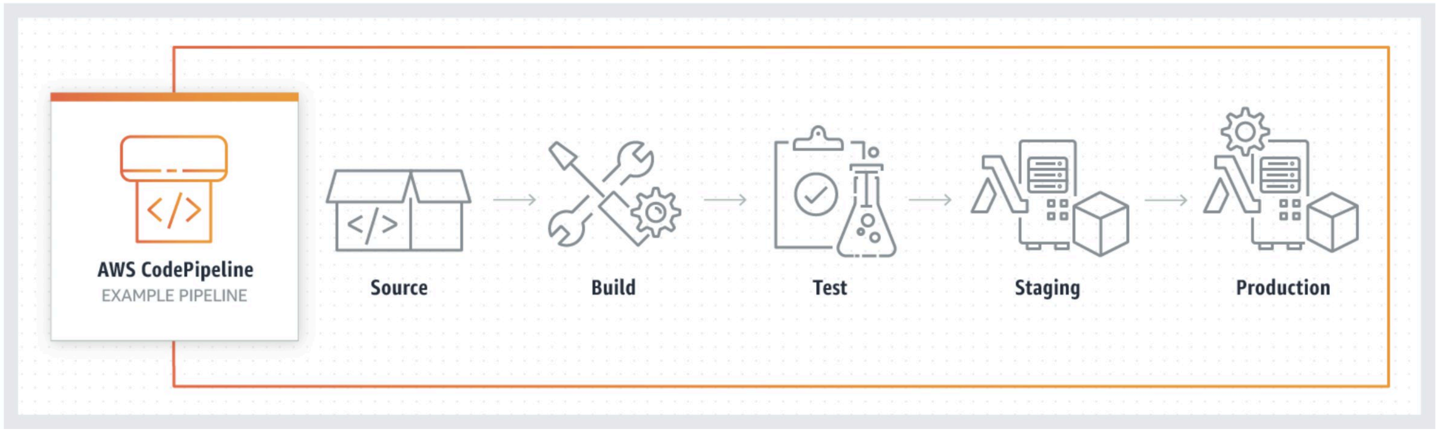
As detailed in the *5G Network Evolution with AWS whitepaper*, IaC is a key driver to automate the provisioning process and life cycle management for both the application and its environment. Rather than relying on manually performed steps, both network/IT administrators and developers can instantiate infrastructure using configuration files. IaC treats these configuration files as software code. These files can be used to produce a set of artifacts: namely the compute, storage, network, and application services that comprise an operating environment. IaC eliminates configuration drift through automation, thereby increasing the speed and agility of infrastructure deployments.

In the case of Network Function Virtualization (NFV) implementation on AWS, this IaC framework brings a value from the perspective of orchestration. From the Virtual Private Cloud (VPC) creation to the network function deployment, every step can be programmed, managed as a source code, and maintained with version control in AWS CodeCommit.

This IaC framework for network function results in repeatable and reliable infrastructure and network function creation and deployment, which can be stretched to the end-to-end (E2E) automation of network slice management and service lifecycle management. AWS provides a comprehensive toolset for creating, maintaining, and deploying infrastructure in a programmatic, descriptive, and declarative way, using services such as AWS CloudFormation, AWS CDK, AWS CDK for Kubernetes, and API exposure of all AWS Services.

CI/CD on AWS

CI/CD can be pictured as a pipeline, where new code is submitted on one end, tested over a series of stages (source, build, test, staging, and production), and then published as production-ready code.



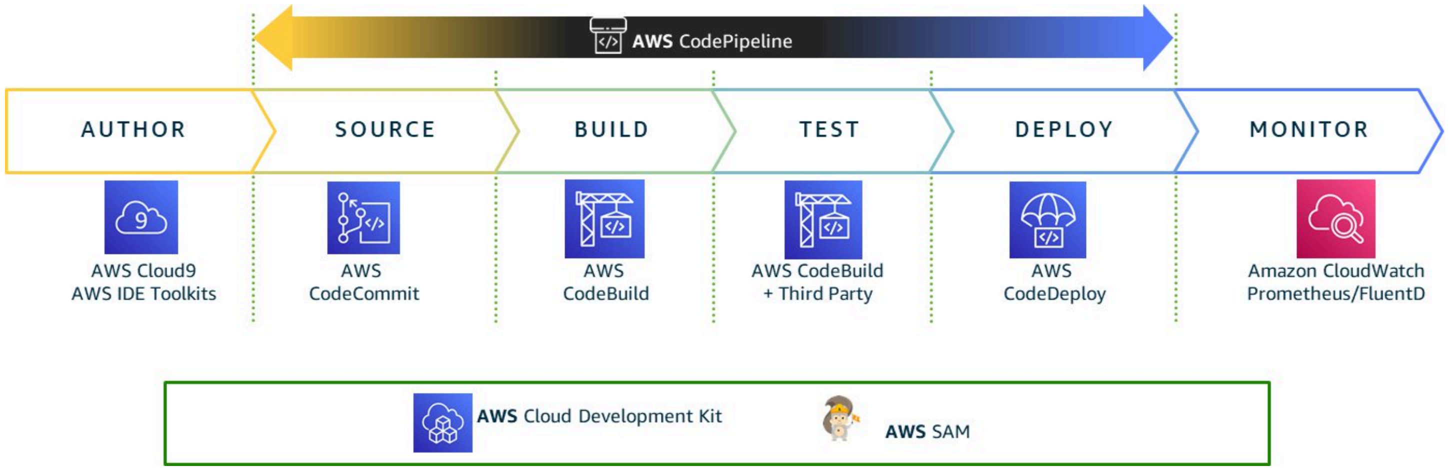
CICD pipeline overview

Each stage of the CI/CD pipeline is structured as a logical unit in the delivery process. Each stage acts as a gate that vets a certain aspect of the code. As the code progresses through the pipeline, the assumption is that the quality of the code is higher in the later stages, because more aspects of it continue to be verified. Problems uncovered in an early stage stop the code from progressing through the pipeline. Results from the tests are immediately sent to the team, and all further builds and releases are stopped if software does not pass the stage.

AWS brings in a complete set of CI/CD developer tools to accelerate software development and release cycles. AWS CodePipeline automates the build, test, and deploy phases of the release process every time there is a code change, based on the defined release model. This enables the rapid and reliable delivery of features and updates.

Code pipelines can integrate with other services. These can be AWS Services, such as Amazon Simple Storage Service (Amazon S3), or third-party products, such as GitHub. AWS CodePipeline can address a variety of development and operation use cases including:

- Compiling, building, and testing code with AWS CodeBuild
- Continuous delivery of container-based applications to the cloud
- Pre-deployment validation of artifacts (such as descriptors and container images) required for network service or specific cloud-native network functions
- Functional, integration, and performance tests for containerized network function/virtual network function (CNF/VNF), including baseline and regression testing
- Reliability and disaster recovery (DR) testing.



AWS CICD pipeline components

AWS can set up CI/CD pipelines using the following AWS Developer Tools:

- AWS CodeCommit
- AWS CodeBuild

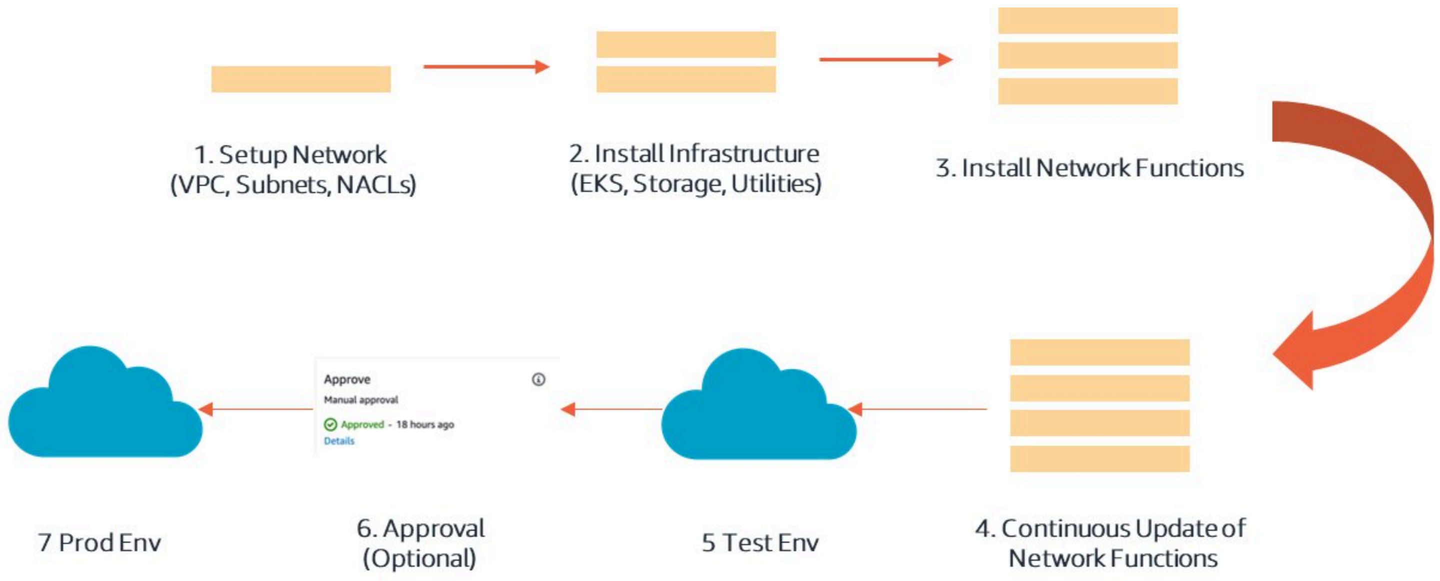
- AWS CodePipeline
- AWS CodeDeploy
- Amazon Elastic Container Registry
- AWS CodeStar

CI/CD pipeline creation can be automated using AWS CDK and AWS CloudFormation. In the NFV domain, this AWS native automation can be integrated into a Management and Orchestration (MANO) framework and the CSP's service orchestration framework.

The CI/CD process includes the following steps:

- Network setup – AWS CDK and AWS CloudFormation initiate creation of the network prerequisites:
  - Networking stack (VPC, subnets, network address translation (NAT) gateway, route table, and internet gateway)
- Infrastructure deployment – AWS CDK and AWS CloudFormation initiate the creation of the following resource stacks:
  - Compute stack (Amazon Elastic Kubernetes Service (Amazon EKS) cluster creation, EKS Worker nodes, AWS Lambda)
  - Storage stack (Amazon S3 buckets, Amazon Elastic Block Store (Amazon EBS) volumes, and Amazon Elastic File System (Amazon EFS)
  - Monitoring stack (CloudWatch , Amazon OpenSearch Service (OpenSearch Service)
  - Security stack (AWS Identity and Access Management (AWS IAM), Amazon Elastic Compute Cloud (Amazon EC2) security groups, VPC network access control lists (NACLs)
- **Cloud Network Function (CNF) deployment** – In this stage, CNF is deployed onto EKS clusters using Kubectl and Helm charts tools. This stage also deploys any specific applications or tools needed by the CNFs to work efficiently (such as Prometheus or Fluentd). CNFs can either be deployed via Lambda functions or with AWS CodeBuild.

- Continuous updates and deployment – These are a sequence of steps that are carried out iteratively to deploy changes that are part of container/configuration changes resulting in upgrades. Similar to the CNF deployment case, continuous updates and deployment can be automated using AWS Services, with the trigger from AWS CodeCommit, Amazon Elastic Container Registry (Amazon ECR), or a third-party source system such as GitLab Webhooks.


1. Setup Network (VPC, Subnets, NACLs)
2. Install Infrastructure (EKS, Storage, Utilities)
3. Install Network Functions
4. Continuous Update of Network Functions
5 Test Env
Approve
Manual approval
Approved - 18 hours ago
Details
6. Approval (Optional)
7 Prod Env

AWS CICD pipeline flow diagram

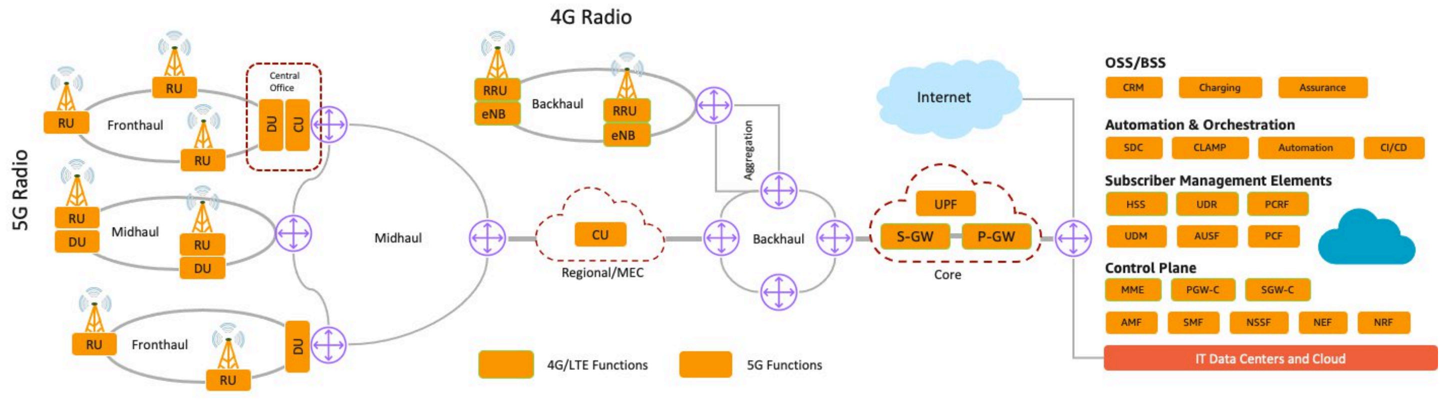
The CI/CD pipeline is built using AWS CodePipeline, and utilizes a continuous delivery service that models, visualizes, and automates the steps required to release software. By defining stages in a pipeline, you can retrieve code from a source code repository, build that source code into a releasable artifact, test the artifact, and deploy it to production. Only code that successfully passes through all these stages will be deployed. You can optionally add other requirements to your pipeline, such as manual approvals, to help ensure that only approved changes are deployed to production.

5G networks on AWS

The typical model of 5G network infrastructure is composed of a 4G/5G radio site, a fronthaul/midhaul/backhaul network, a core network site, and a telecom/IT data center. CSPs can use AWS Services to create a scalable, flexible 5G network infrastructure while reducing upfront investment cost. AWS can be used to implement the virtual Network Operation Center (NOC) in the Region that hosts the Operations Support System/Business Support System (OSS/BSS) and the majority of control plane core network functions.

AWS can also be leveraged. to implement the local central office (CO) or distributed data center with a fleet of AWS Outposts instances that host mostly user-plane functions such as UPF (user-plane function), RAN central unit (CU), and Multi-Access Edge Computing (MEC). A more detailed explanation of the reference architecture and the benefits of 5G network implementation on AWS is explained in the *5G Network Evolution on AWS* whitepaper.

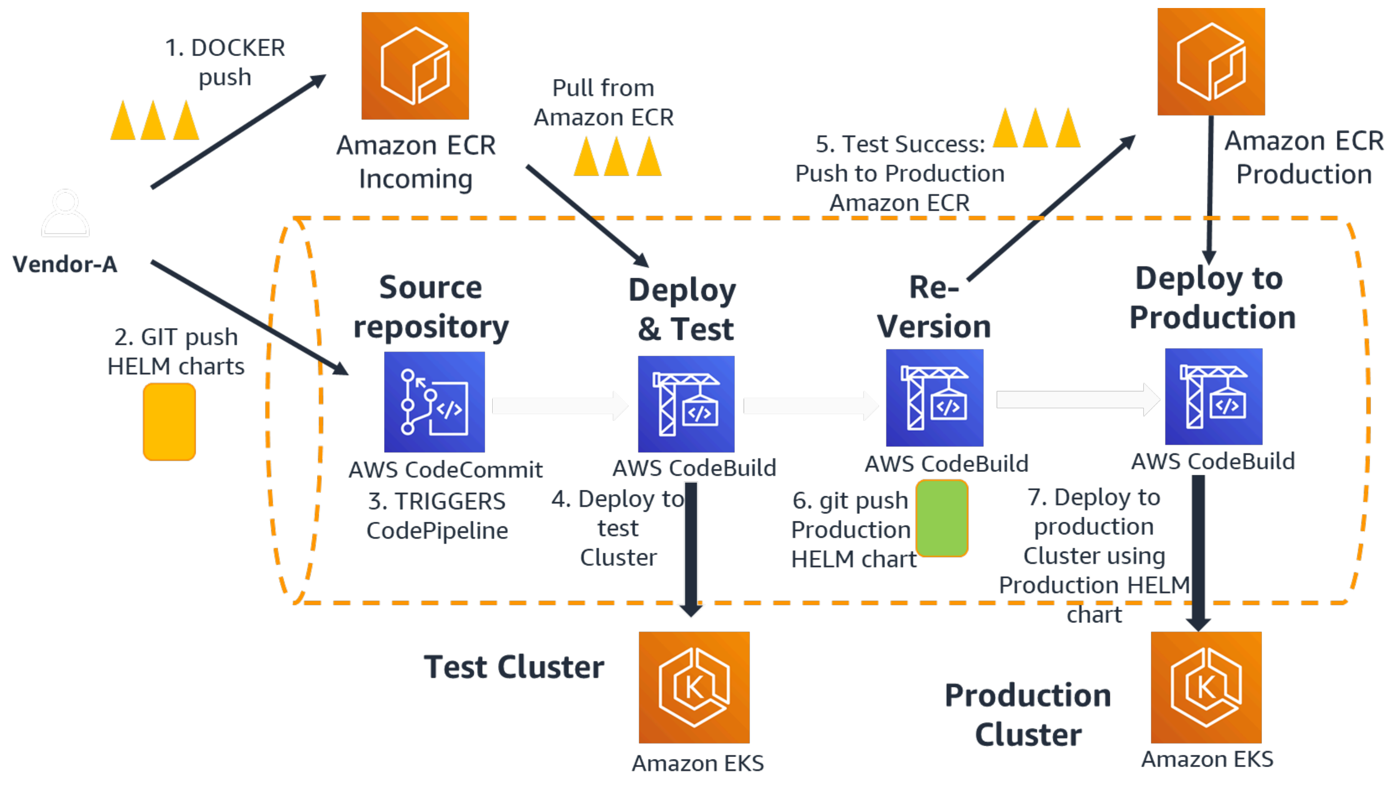
When it's time to implement the 5G network on AWS, AWS CI/CD tools, which are introduced in the following sections of this whitepaper, can facilitate the full automation of deployment, upgrade, and lifecycle management of 5G network functions.



5G network E2E architecture

CI/CD in 5G networks

The design construct of the infrastructure is stored in the form of code using declarative language. This enables the CSP to have a repeatable reproduction of the infrastructure with the same expected behavior as needed. The code is maintained in the code repository, and a pipeline is set up to orchestrate updates to the deployed stacks (for example, AWS CDK and AWS CloudFormation). AWS can help build Infrastructure as Code (IaC) for agile onboarding of Independent Software Vendor (ISV) functions.



Code pipeline flow

Changes in cloud-native network function configurations through Helm charts are considered to be triggers for an automatic CI/CD pipeline execution for network functions.

AWS CodeCommit can be used to maintain configuration files, and Amazon ECR can be used to preserve container images.

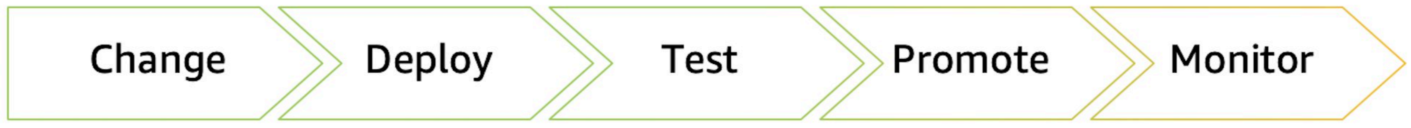
As shown in the *Code pipeline flow* figure, when the ISV pushes new code changes into the code repository (Helm chart, config files, or a properties file), the code pipeline is triggered. The code pipeline pulls the image from ECR and use the Helm chart to deploy the application. The new application testing can be integrated with the third-party test automation framework. Based on the result, CSPs can approve for production deployment.

The CodePipeline source stage looks for changes in configuration files. The valid providers for source stage are CodeCommit, Amazon S3, GitHub, or AWS CloudFormation. Alternative source systems can be integrated by using Lambda functions to implement Webhooks, which enables event-driven integration between Gitlab and AWS CodePipeline. See the following links for a detailed implementation guide.

- Webhooks with GitLab

- Container registry integrations

CI/CD pipeline design should account for critical deployment steps such as initial deployment, testing, and promotion to production after test results are aligned with expectations and verified against the baseline. Every stage of the pipeline process provides data artifacts, which enable comparison and data-driven decisions.



Application CI/CD pipeline steps

Every stage can be considered a separate task, allowing the incorporation of validation and deployment workflows adequate to support network service and cloud-native network functions. Run tasks can incorporate additional third-party tools such as traffic generators and simulators, enabling end-to-end network service validation.

AWS provides a sophisticated AWS Step Function (cloud-native state machine) service that natively integrates with other AWS Services, and also has the ability to integrate with external systems such as Jira or a test automation framework.

CI/CD detailed steps

CI/CD can be pictured as a pipeline, where new code is submitted on one end, tested over a series of stages (source, build, test, staging, and production), and then published as production-ready code.

Here are the steps of deployment and testing. The deployment and configuration are primarily split in four main sections:

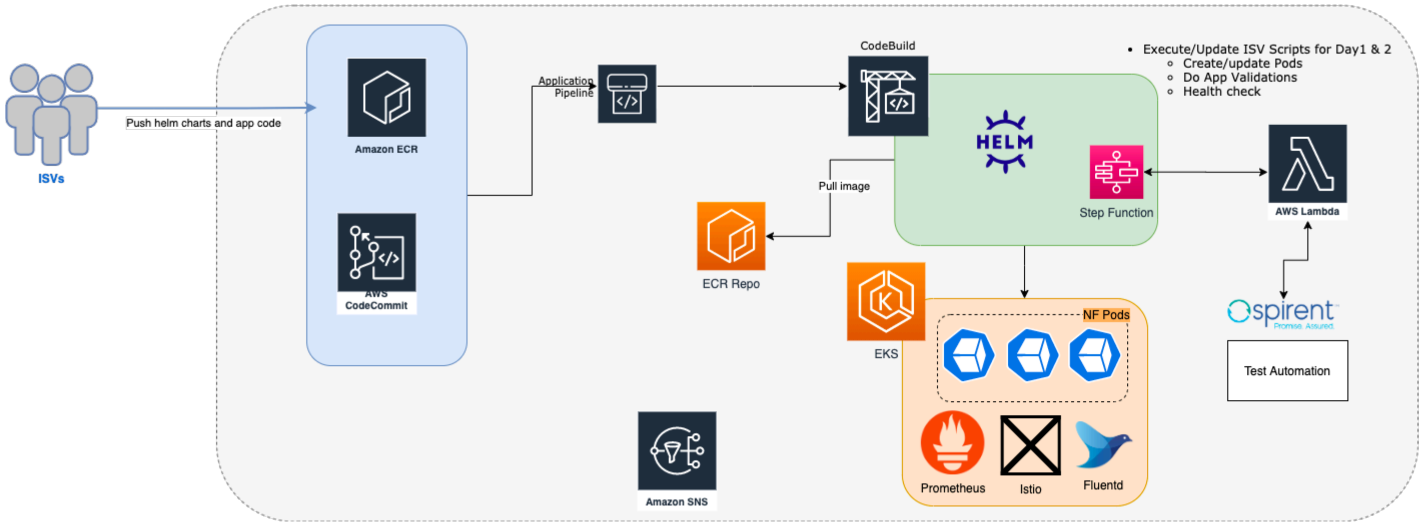
- Network setup
- Infrastructure deployment
- Cloud-native network function deployment
- CNF continuous delivery

Network setup

Focus on the setup of the infrastructure prerequisites. This includes the creation of the VPC, networks, subnets, NACLs, and so on. Design the IP network plan considering ISVs and the customer's implementation plan (such as multi tenancy and static vs. dynamic allocations). This plan can be coded in AWS CDK or AWS CloudFormation. Running this code deploys cloud infrastructure network prerequisites.

Infrastructure deployment

Infrastructure deployment provisions any infrastructure components. It includes spawning the EKS cluster and the supporting infrastructure, such as EFS, EKS Worker nodes, ELBs, and configuring the cluster according to cloud-native network function requirements. Based on the CNF requirements, AWS also deploys extra network interfaces for the nodes, including Multus interfaces. Most of the deployment and configuration steps are a one-time effort for an application, and are updated only when needed as an update for the application.

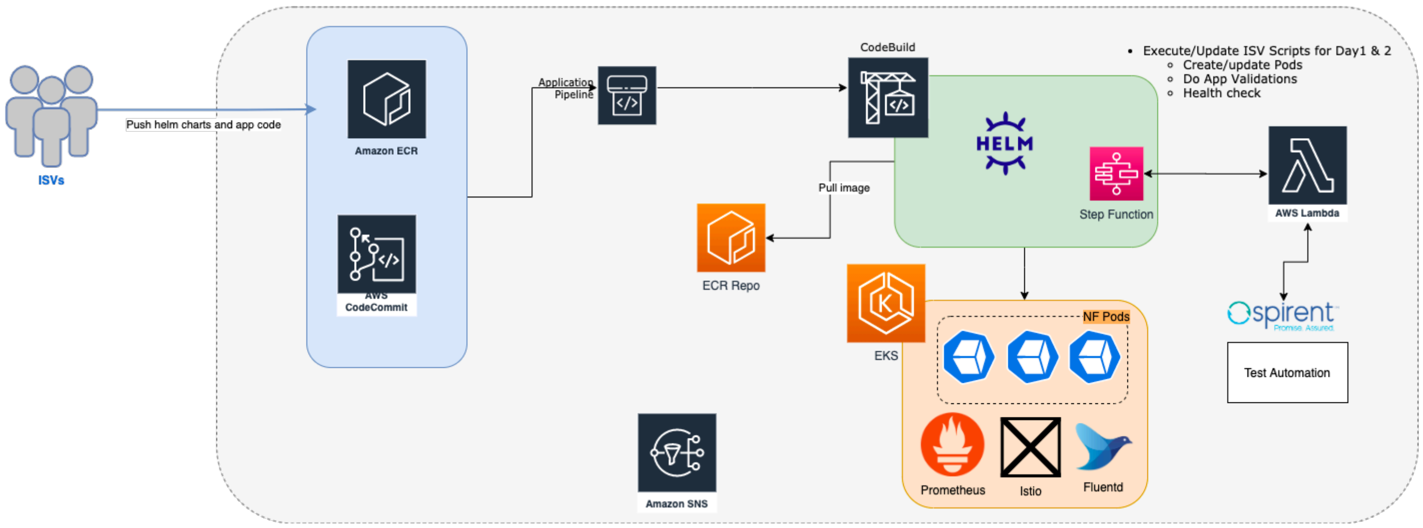


Infrastructure deployment with CDK

Cloud-native network function deployment

CNF deployment is about the application deployment. As part of the CNF deployment, the Helm charts for the application are implemented through the CI/CD code pipeline. The callback to run individual application-specific scripts primarily involving pre- and post-checks are incorporated. The Helm charts are implemented in a sequence as per the application's needs, and check for the status of the Kubernetes PODs before moving to the next step in the deployment. Often, ISVs provide a wrapper script to run the Helm charts and sanity checks. These ISV scripts are invoked from within AWS CodePipeline. As part of this phase, the logging and monitoring agents such as Prometheus and Fluentd are deployed in addition to Amazon CloudWatch, which logs and monitors the cloud infrastructure of the application.

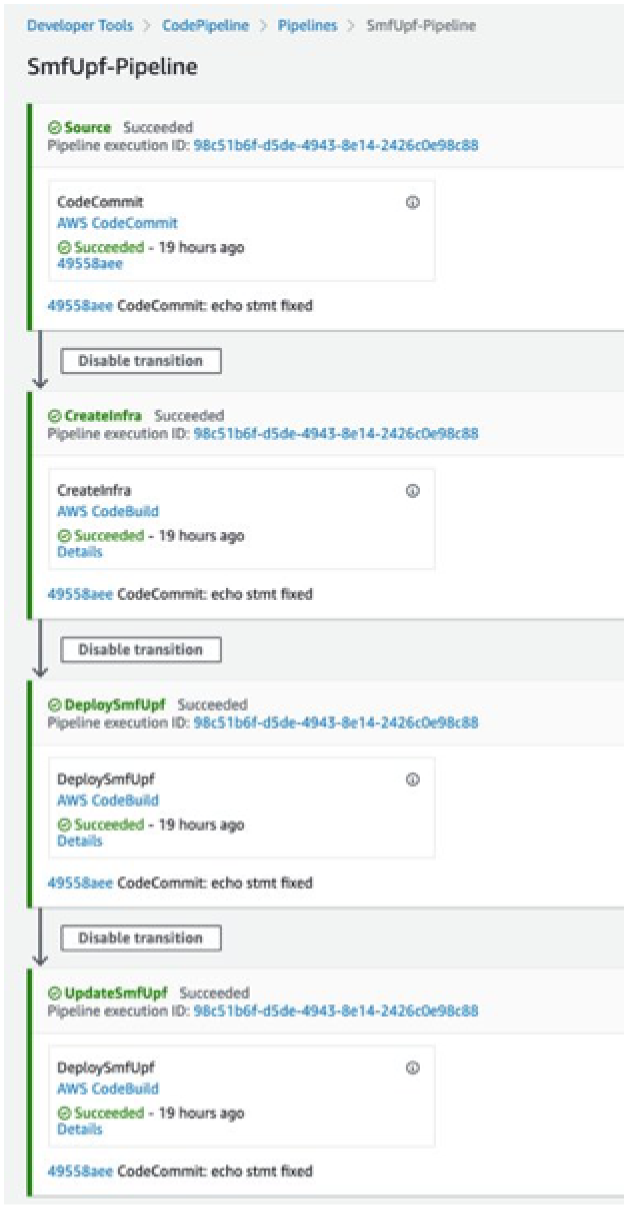
The code pipeline is integrated with the third-party test automation framework. The code pipeline can directly call the test automation framework APIs to run the test on the deployed application, query the test results, and analyze the result. This simplifies the deployment and testing of the application.



Application deployment and update

The following is an example of the deployment of the user plane function/session management function (UPF/SMF) CNF via AWS CodePipeline.

- Automate the complete CI/CD process using CodeCommit, CodeBuild, and CodePipeline.
- Infra creation and application installation tasks are integrated as part of the pipeline.
- FluentD and Prometheus agents are installed and created in Amazon CloudWatch dashboards.


Developer Tools > CodePipeline > Pipelines > SmfUpf-Pipeline

SmfUpf-Pipeline

Source Succeeded
Pipeline execution ID: 98c51b6f-d5de-4943-8e14-2426c0e98c88

CodeCommit
AWS CodeCommit
Succeeded - 19 hours ago
49558aee

49558aee CodeCommit: echo stmt fixed

CreateInfra Succeeded
Pipeline execution ID: 98c51b6f-d5de-4943-8e14-2426c0e98c88

CreateInfra
AWS CodeBuild
Succeeded - 19 hours ago
Details

49558aee CodeCommit: echo stmt fixed

DeploySmfUpf Succeeded
Pipeline execution ID: 98c51b6f-d5de-4943-8e14-2426c0e98c88

DeploySmfUpf
AWS CodeBuild
Succeeded - 19 hours ago
Details

49558aee CodeCommit: echo stmt fixed

UpdateSmfUpf Succeeded
Pipeline execution ID: 98c51b6f-d5de-4943-8e14-2426c0e98c88

DeploySmfUpf
AWS CodeBuild
Succeeded - 19 hours ago
Details

49558aee CodeCommit: echo stmt fixed

Deployment example of UPF/SMF CNFs

CNF continuous delivery

This step consists of a sequence of steps that are carried out repeatedly to deploy changes that are part of container/configuration changes resulting in upgrades. The CNF continuous delivery is automated via pipelines and is specific to individual applications. AWS uses standard Helm charts to update specific CNFs. The code pipeline has pre- and post-checks for the application update status. The updated CI/CD pipeline is also integrated with a test automation framework to run automated testing. This abstraction allows a clean deployment of network functions.

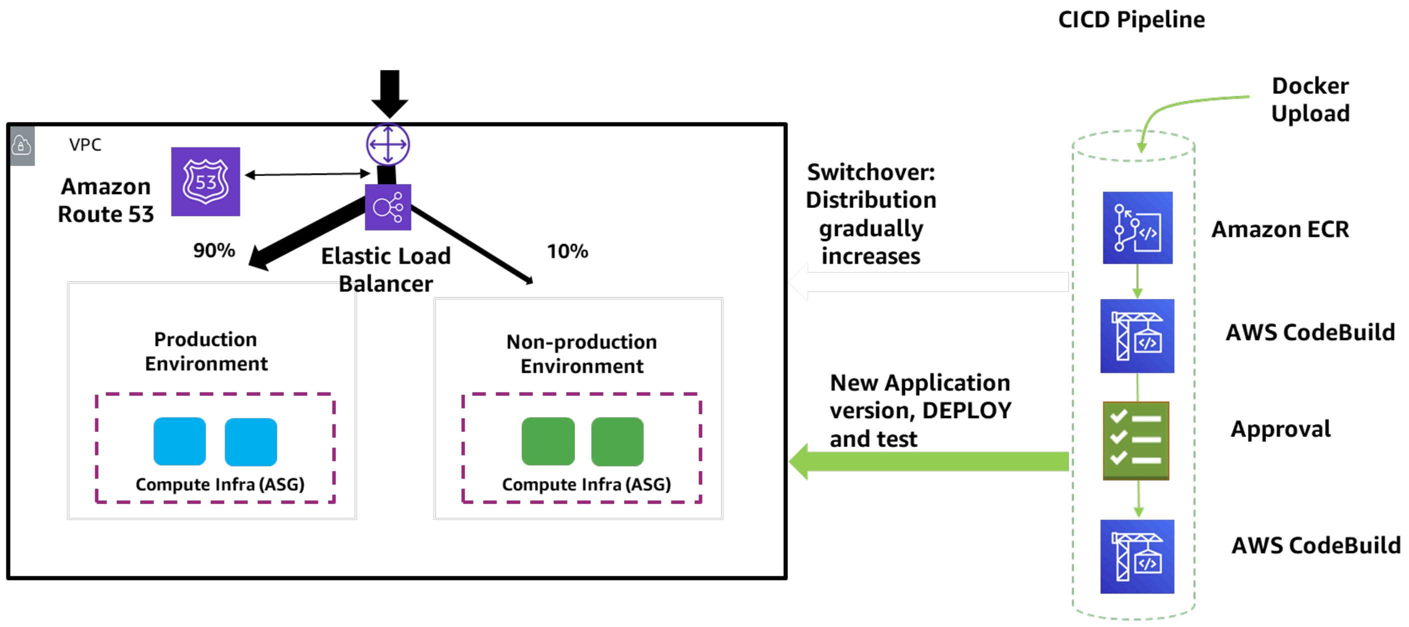
CNF continuous delivery and deployment can be broadly classified into the following categories:

- **Application upgrades** — Most application upgrades are changes within the Kuberbetes application PODs. These updates can be applied automatically through the code pipeline. Most CNFs support in-place upgrades by providing multiple instances of application PODs. Multiple instances allow a rolling upgrade approach. Not all application POD changes support Helm upgrade. Pipelines take these variations into account and use Helm install/delete as required.
- **Major upgrades** — Major upgrades are primarily database schema changes. This change cannot be applied without causing some downtime. The standard approach to these changes is to delete the application and re-create the relevant pods. During the process the application may not be available. The following tools are used for upgrades:

  - **AWS CloudFormation** enables customers to describe and provision all the infrastructure resources in JSON or YAML templates. AWS CloudFormation provides a powerful extension mechanism through Lambda-backed custom resources. Customers can extend AWS CloudFormation beyond AWS resources, and provision the required resource in other environments; for example, on-premises resources in hybrid environments. AWS CDK offers developers the ability to build out code using higher level familiar programming languages such as Python, TypeScript, JavaScript, Java, and C#, and then compile the code into a lower-level AWS CloudFormation JSON format, which can then be deployed.
  - **BlueGreen deployment** — AWS supports and recommends blue/green and canary-based deployments in test as well as in production environments. Blue/green deployments enable customers to test a new application version in a contained environment. They provide an easy and graceful method to switch over production traffic. Canary-based deployments extend this concept by enabling the non-production green environment to be tested with a small proportion of production traffic to uncover any issues caused by production traffic. The new application version is tested against internal simulated test traffic as well as small amounts of production traffic, which gives the user confidence before they switch over the production traffic. The production traffic is gradually increased until switchover is complete. Implementation involves weighted DNS and weighted ELB target groups.
  - **Automation** can be achieved by configuring AWS CodePipeline with the blue/green and canary-based deployment stages. The approval stage may be manually driven initially during provisioning, but later it should be fully automated. In the test environments, it is good practice to always test with a rollback action to validate

forward and backward compatibility, before deploying into production. The blue/green deployment on clusters with service mesh depend on the support provided by the end-application and the routing gateway for the service mesh to accomplish a graceful transition.

- **AWS Systems Manager** provides a unified user interface so you can view operational data from multiple AWS Services used by network functions deployed by CI/CD. Systems Manager enables you to automate operational tasks across your AWS resources.



Canary deployment

Security

Security is a critical element. Following is a list of security steps that the AWS CI/CD process takes into account when deploying an application.

- **Source** — The ECR repository allocated to the vendor is configured with a "Scan on Push" flag enabled, so that any uploads of Docker images will be immediately subjected to a security scan. Any known common vulnerabilities and exposures (CVE) will be flagged with notifications. Apart from ECR, when vendors put charts into the AWS CodeCommit repository, they are requested to encrypt any passwords used with Secrets Manager rather than with plain text.

- **Artifacts integrity** — The artifacts used across the pipeline are encrypted whether at rest (using AWS managed keys) or transit (using SSL/TLS).
- **Users and roles** — The permissions provided to the users or resources are based on the principle of least privilege. There should be a cross-role trust relationship that may need to be configured if you are operating across resources in different services. For example, AWS CodeBuild needs permission to run commands on an Amazon EKS cluster.
- **Audit** — The auditing capability offered by AWS CloudTrailtracks each and every API call across services and user operations, and enables the evaluation of past events.
- **Image vulnerability scanning** — CNF images that are uploaded to Amazon ECR are automatically scanned for security vulnerabilities. A report of the scan findings is available in the AWS Management Console, and can also be retrieved via API. The findings can then be sent to CSP operators for corrective action, including replacement of the CNF image.

Security checks occur at various stages of the pipeline to ensure that the newly uploaded image is secure and complies with the desired compliance checks so a notification can be sent to CSPs for approval:

- The container registry scans for any open CVE vulnerabilities.
- The configuration is checked for information leaks, known personally identifiable information (PII) patterns, during the test stage, triggering compliance check rules for problems such as unexpected open TCP/UDP ports and DOS vulnerabilities.
- Backward and forward compatibility is verified for upgrade/rollback safety.

Apart from the application, it is critical to provision pipeline security by ensuring the encrypted transfer of artifacts across stages, whether at rest or in transit.

Observability

AWS enables observability for the 5G CNFs that are deployed on AWS by default. This is enabled by Amazon CloudWatch. CloudWatch brings complete visibility to your cloud resources and applications.

Amazon CloudWatch has four major steps during this process:

1. **Collect** — Collect metrics and logs from all of your AWS resources, applications, and services that run on AWS and on-premises servers.

2. **Monitor**— Visualize applications and infrastructure with CloudWatch dashboards, correlate logs and metrics side by side to troubleshoot, and set alerts with CloudWatch Alarms .
3. **Act** — Automate response to operational changes with CloudWatch Events and AWS Auto Scaling.
4. **Analyze** — Up to one-second metrics, extended data retention (15 months), and real-time analysis with CloudWatch Metric Math .

The Amazon CloudWatch agent is installed in the customer’s Kubernetes cluster. The agent supports Prometheus configuration, discovery, and metric pull features, enriching and publishing all high fidelity Prometheus metrics and metadata as Embedded Metric Format (EMF) to CloudWatch Logs.

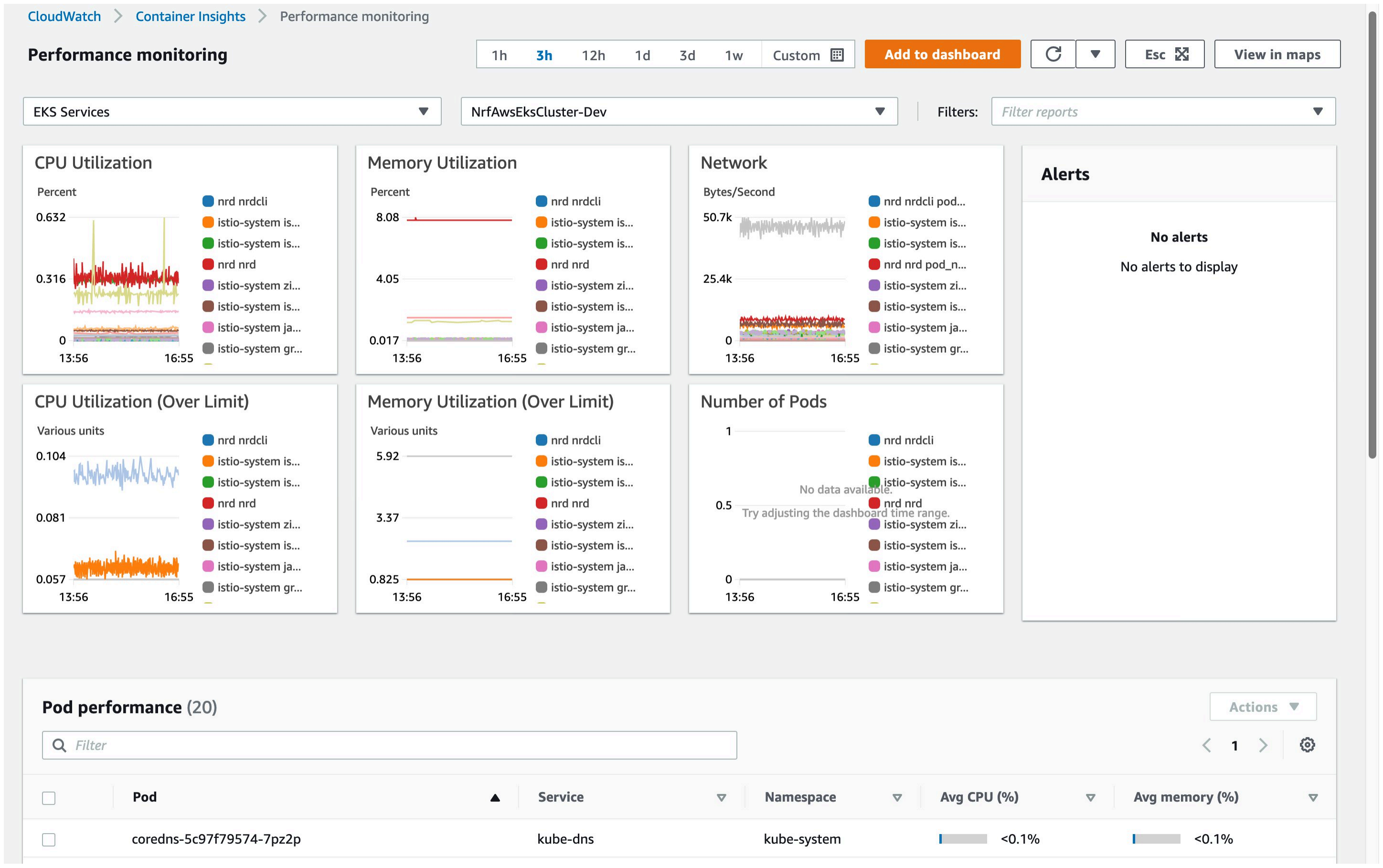
Amazon CloudWatch Container Insights automates the discovery and collection of Prometheus metrics from containerized applications. It automatically collects, filters, and creates aggregated custom CloudWatch metrics visualized in dashboards.

Each event creates metric data points as CloudWatch custom metrics for a curated set of metric dimensions that is fully configurable. Publishing aggregated Prometheus metrics as CloudWatch custom metrics statistics reduces the number of metrics needed to monitor, alarm, and troubleshoot performance problems and failures. You can also analyze the high-fidelity Prometheus metrics using CloudWatch Logs Insights query language to isolate specific pods and labels impacting the health and performance of your containerized environments.

AWS CloudTrail offers this visibility, recording every API call across services. AWS Config offers capability for compliance validation. AWS provides customers with additional monitoring options of metrics, logs, events for the application, infrastructure, and pipelines, using various services like AWS X-Ray and AWS CloudTrail.

- AWS can natively integrate open-source metric tools like Prometheus, Fluentd, and so on.
- Prometheus metrics can be further ingested into Amazon CloudWatch or OpenSearch Service for further analysis.
- AWS uses fluentD as a standard mechanism to collect logs from various systems. That same mechanism is used and configured for this project.

For details on how to configure this mechanism, see Set Up FluentD as a DaemonSet to Send Logs to CloudWatch Logs.



Pod performance (20)

Pod	Service	Namespace	Avg CPU (%)	Avg memory (%)
coredns-5c97f79574-7pz2p	kube-dns	kube-system	<0.1%	<0.1%

Example of Amazon CloudWatch monitored metrics

CI/CD orchestration with third party and open-source tools

The orchestration layer uses IaC to deploy and configure the underlying infrastructure required to run the 5G network functions. This layer should be designed to be modular, portable and reusable.

The infrastructure follows cloud-native best practices, being highly-available, redundant, and scalable.

As demonstrated in the previous sections, the deployment of the underlining infrastructure can be achieved using the AWS Cloud Development Kit (AWS CDK). This can be accomplished using Terraform by Hashicorp.

Terraform

Terraform is an IaC software tool that provides a consistent command line interface (CLI) workflow to manage hundreds of cloud services. Terraform codifies cloud APIs into declarative configuration files.

For deployment with Terraform, use the same principles used in CDK. The code is structured in modules that allow the networking components to be customized and reused according to the vendor requirements.

The configuration is all parameterized, which allows the deployments to be fully tailored according to providers and ISV recommendations.

The network functions deployment is separated in two phases:

- The required AWS infrastructure is created and managed via a central repository.
- The configuration and code is centrally stored in a GitHub repository.

After the prerequisites are created, the network function is ready to be deployed by using an application pipeline that was set in the previous stage.

Infrastructure deployment

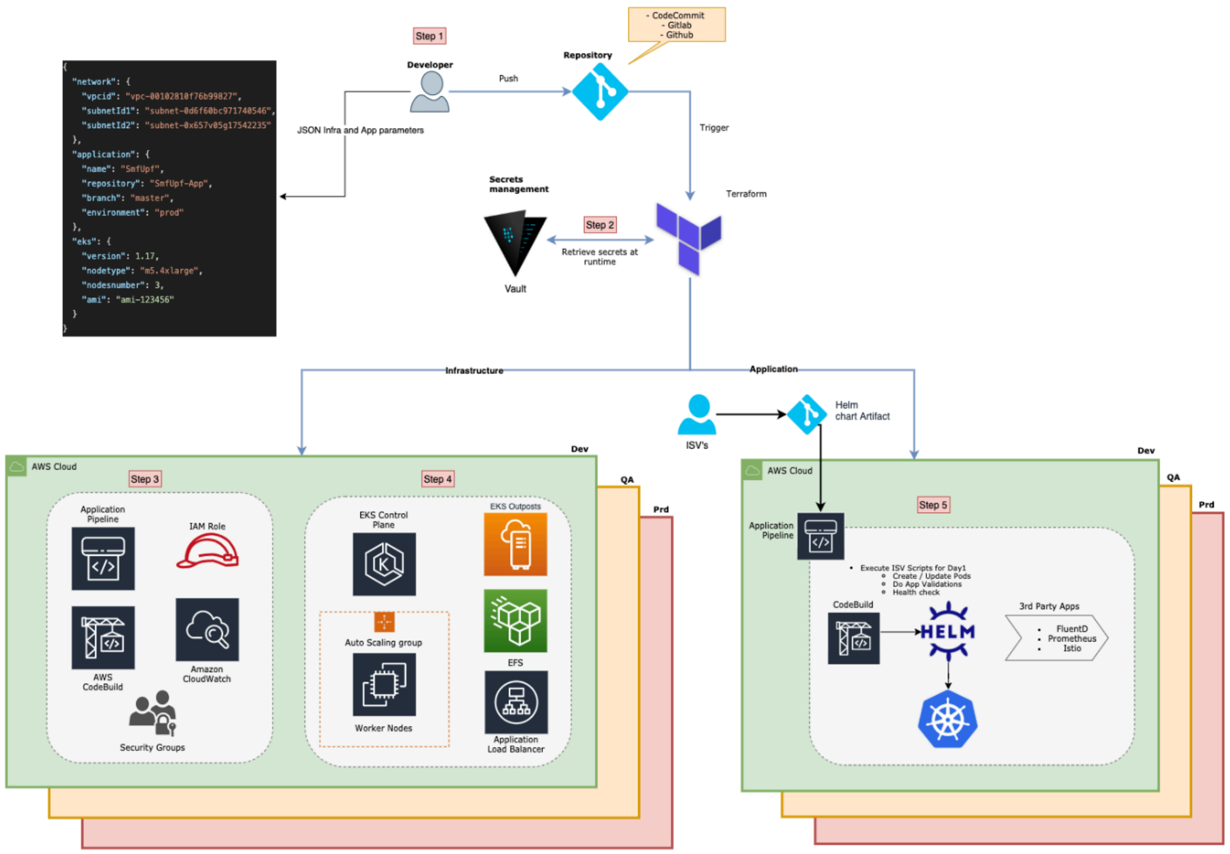
Infrastructure deployment includes all the prerequisites for the network function to be successfully deployed and configured.

Some of the components created as part of this phase are:

- Networking — VPC, public and private subnets, routes, load balancers
- Compute — Kubernetes (Vmware Tanzu , Amazon EKS, or AWS Outposts), Amazon EC2 instances primary and worker nodes, auto scaling group
- Storage — Amazon EFS, Amazon EBS, Amazon S3 bucket
- Security — Security groups
- Pipeline — CodePipeline, CodeBuild
- Observability — CloudWatch, Prometheus, FluentD

Here is the infrastructure sequence orchestrated by Terraform and explained in the following figure:

1. A developer populates a JSON file that is stored in a central repository with the IaC code. The file contains information about the desired infrastructure configuration such as instances size, Kubernetes version, network information, and application repository details.
2. Retrieves secrets from HashiCorp Vault or AWS Secrets Manager at runtime.
3. Deploys and configures the infrastructure components (networking, compute, storage, and security).
4. An Amazon EKS cluster with worker nodes that hosts the network function pods is deployed. Amazon EKS can also be deployed on AWS Outposts to support workloads that require proximity to a datacenter.
5. An application pipeline is created and configured to listen for changes in the network function repository. Every time code is pushed to the configured repository branch, the pipeline automatically triggers build, test, and deployment of the network function.
6. Observability tools that collect and centralize logs and metrics are deployed as services in all the nodes, and provide almost real-time data that can be visualized in Grafana or OpenSearch Dashboards



```
{
  "network": {
    "vpcid": "vpc-00102810f76b99827",
    "subnetId1": "subnet-0d6f60bc971740546",
    "subnetId2": "subnet-0x657v05g17542235"
  },
  "application": {
    "name": "SmfUpf",
    "repository": "SmfUpf-App",
    "branch": "master",
    "environment": "prod"
  },
  "eks": {
    "version": 1.17,
    "nodetype": "m5.4xlarge",
    "nodesnumber": 3,
    "ami": "ami-123456"
  }
}
```

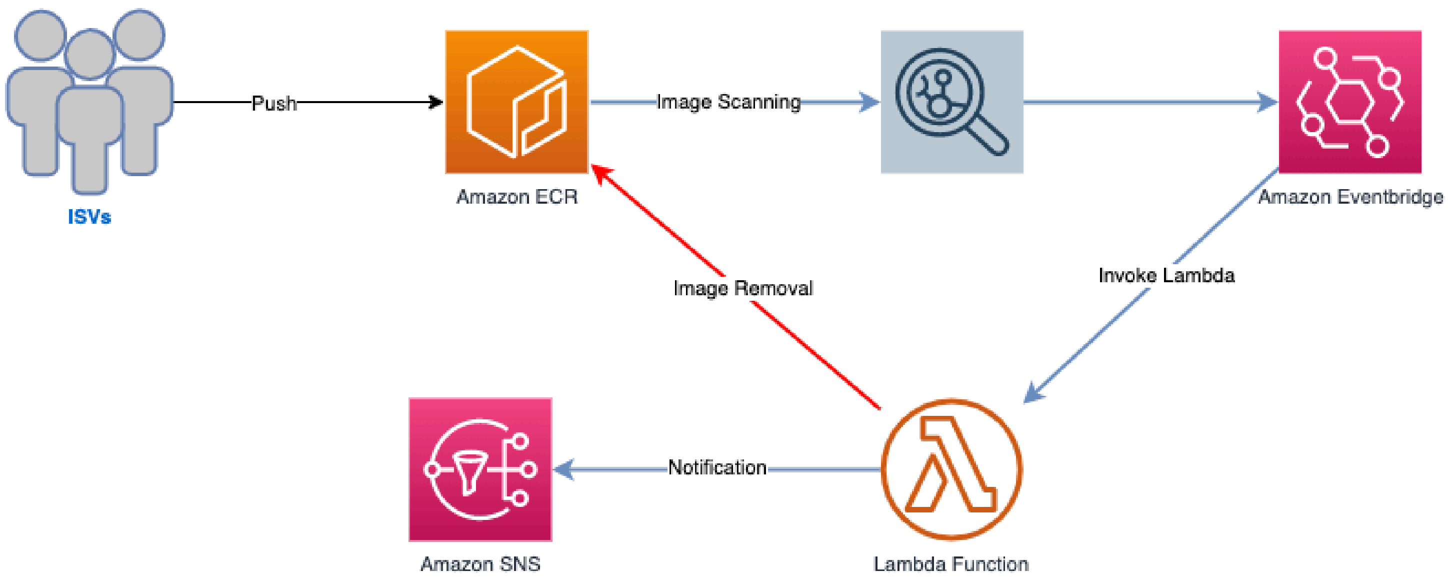
Network function deployment and configuration

Network function deployment and configuration

The pipeline created in the previous stage enables both ISVs and providers to decentralize and optimize the deployment of network functions. The pipeline is connected and listens to changes in the application repository, which is configured in the JSON file in step 1 of the preceding figure.

To vet the images that are published by third parties, a vulnerability scanning solution that assists in identifying software vulnerabilities in container images is deployed and configured. The scanning solution automatically checks all new images pushed to Amazon ECR. For more information about ECR image scanning, see Image scanning.

The following figure demonstrates the architecture of the Image Vulnerability Scanning solution.



Architecture of the Image Vulnerability Scanning solution

The application pipeline can be configured to be triggered by changes in the image after the scanning result, or by direct changes in the repository. For example, when a new Helm image is created.

The following list is the sequence that creates/upgrades the network function, as represented in the following figure:

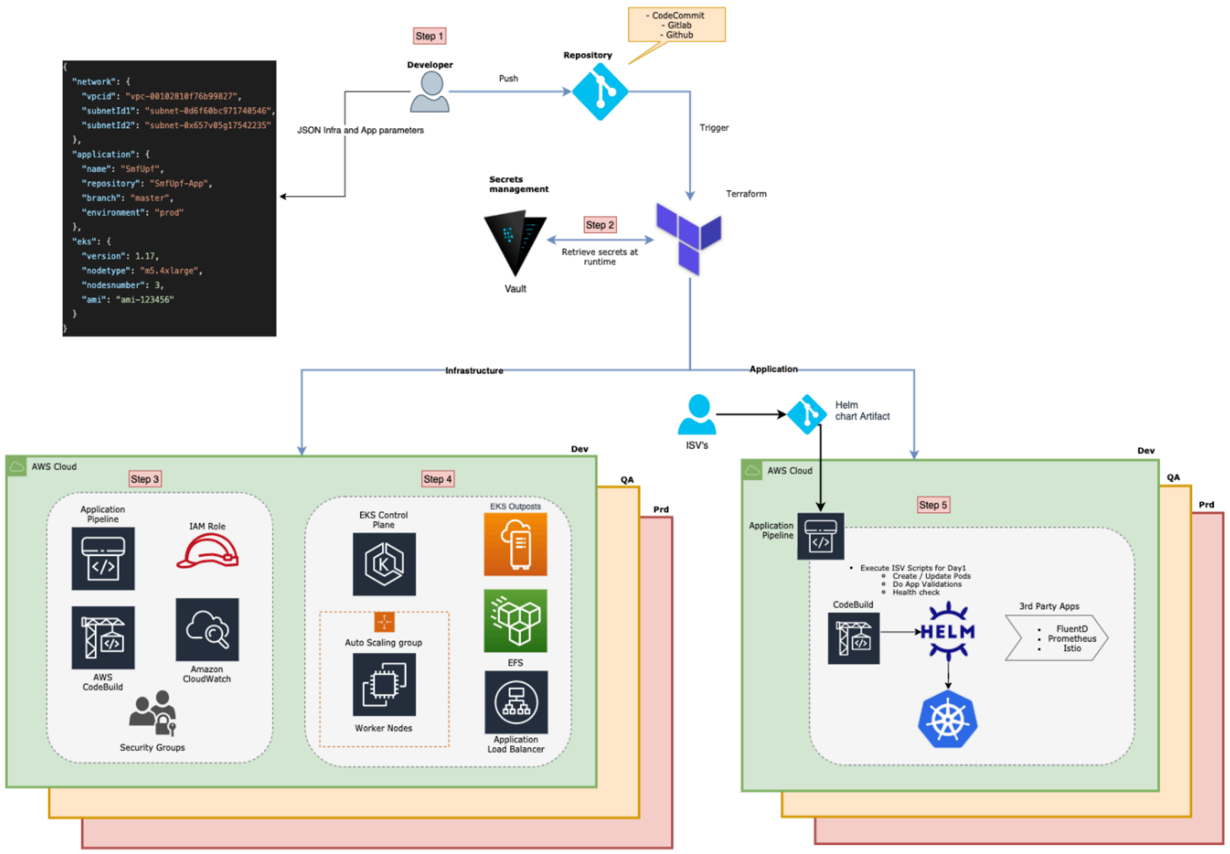
ISVs publish new images to Amazon ECR. (If the image is approved, the application pipeline is triggered.

CodePipeline pulls the new image from Amazon ECR and uses CodeBuild to deploy the image to Kubernetes. Helm commands can be used to upgrade the network function.

After the image is deployed, Test as Service (TaS) is triggered. TaS validates the new deployment and centralizes data and metrics about the network functions' performance under stress.

Logs and metrics are collected and centralized in OpenSearch and Grafana. Third parties such as Datadog, Istio, and Prometheus can also be configured to provide additional observability.

A MANO that can coordinate network resources can also be deployed and integrated with the solution. It uses the data collected to take automated actions such as network slicing and Quality of Service (QoS) auto scaling.



```
{
  "network": {
    "vpcid": "vpc-00102810f76b99827",
    "subnetId1": "subnet-0d6f60bc971740546",
    "subnetId2": "subnet-0x657v05g17542235"
  },
  "application": {
    "name": "SmfUpf",
    "repository": "SmfUpf-App",
    "branch": "master",
    "environment": "prod"
  },
  "eks": {
    "version": 1.17,
    "nodetype": "m5.4xlarge",
    "nodesnumber": 3,
    "ami": "ami-123456"
  }
}
```

Application pipeline

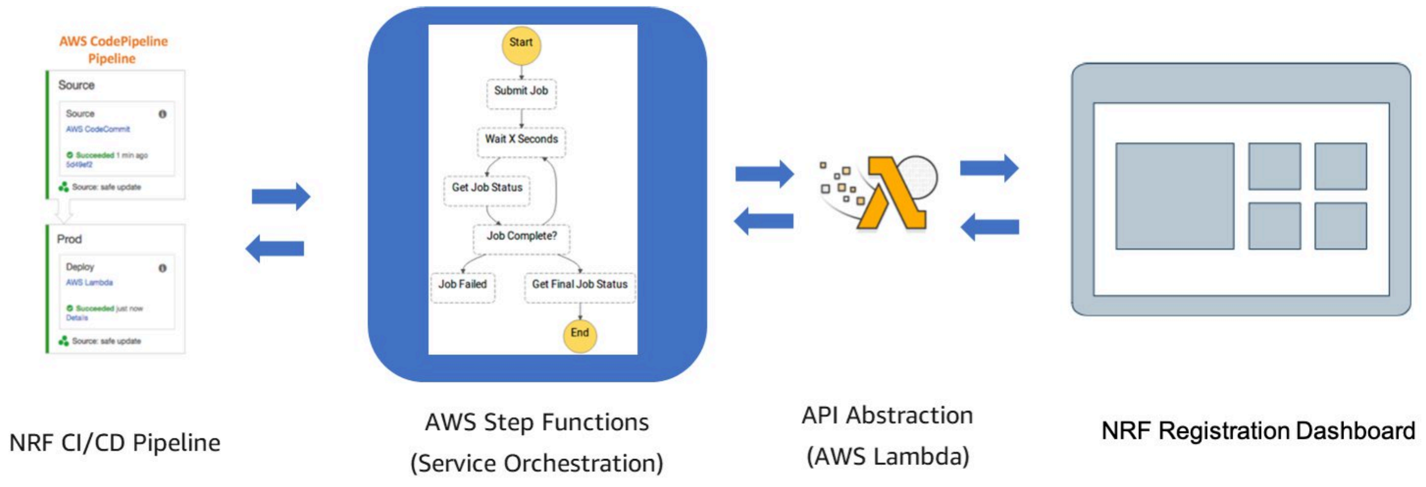
Testing

The telecom-specific test automation framework can be integrated in the code pipeline. The code pipeline is integrated with step functions to orchestrate the integration with a test automation framework. AWS step functions use multiple Lambda functions to invoke the test automation framework (third-party tools) via API calls. The step function will initially get the test ID, run the test, and get the results from the test automation framework. The step function then analyzes the results and passes them to the code pipeline for approval of the application for production deployment. The approval can be automated or maintained manual in the code pipeline as needed. This is an important step for CSPs to promote the deployment from test environment to production. High level APIs required for integration are categorized into the following manner:

- Get context
- Run specific test case
- Stop test case

- Get results

The complexity of calling external REST APIs is modeled using AWS step functions, which allow standard constructs to invoke parallel flows, wait for results, branch based on conditions, and integrate REST API with AWS CodePipeline.



Testing flow

CI/CD and orchestration

Continuous Integration and Continuous Delivery are part of an overall automation philosophy that comes with the cloud-native architecture and how it applies to 5G. Orchestration is another aspect of this philosophy that is expected to be dynamic and reactive to any changes that happen on the network. Orchestration and CI/CD are expected to be tightly coupled to guarantee a healthy service and minimize service disruption. The integration between CI/CD and orchestration is expected to be on two fronts:

- Applying patches and upgrades into the system need to be managed and orchestrated in a way that minimizes disruption to any live services. For example, orchestration can dynamically determine the best time an update should be rolled out.
- CI/CD-aware orchestration allows traffic to be shifted during deployment of upgrades based on the deployment model strategy adopted (canary, linear, or all-at-once).

Typically, orchestration solutions run on top of CI/CD pipelines to allow orchestration to introduce governance phases into those pipelines and to have exposure into ongoing upgrade cycles.

Conclusion

CI/CD provides a clear and efficient path for developers and application teams to deploy new application code in a matter of minutes. AWS has a wealth of tools that can help developers in the integration, testing, and deployment of new code, including AWS CodePipeline, AWS CodeCommit, AWS CodeBuild, AWS CodeDeploy, and many others. This document looked at how AWS services can be used to create a CI/CD process for deploying 5G network function in a fully automated way, including the different steps needed to complete the deployment of new code. It also looked at how to integrate a third-party test automation framework into the CI/CD process, as well as using third-party tools such as Terraform.

Contributors

Contributors to this document include:

- Hisham Elshaer, Senior Consultant, AWS Telecom, Amazon Web Services
- Vara Prasad Talari, Principal Consultant, AWS Telecom, Amazon Web Services
- Rabi Abdel, Principal Consultant, AWS Telecom, Amazon Web Services
- Franco Bontorin, Senior Consultant, Shared Delivery, Amazon Web Services
- Pragtideep Singh, Consultant, Shared Delivery, Amazon Web Services
- Subbarao Duggisetty, Cloud Infra Architect, Global Accounts, Amazon Web Services
- Young Jung, Senior Partner Solutions Architect, AWS Telecom, Amazon Web Services

Document revisions

To be notified about updates to this whitepaper, subscribe to the RSS feed.

Change	Description	Date
Initial publication	Whitepaper first published	March 8, 2021

Further reading

For additional information, see:

- *Practicing Continuous Integration and Continuous Delivery on AWS* (whitepaper)
- *Carrier-Grade Mobile Packet Core Network on AWS* (whitepaper)
- *5G Network Evolution with AWS* (whitepaper)

Acronyms

- **AMF** — Access & Mobility Management Function
- **API** — Application Programming Interface
- **AUSF** — Authentication Server Function
- **BSS** — Business Support System
- **CDK** — Cloud Development Kit
- **CI/CD** — Continuous Integration and Continuous Delivery
- **CLI** — Command Line Interface
- **CNF** — Cloud-native or Containerized Network Function
- **CSP** — Communication Service Provider
- **CU** — RAN Central Unit
- **CVE** — Common Vulnerabilities and Exposures
- **DoS** — Denial of Service
- **DR** — disaster recovery
- **DU** — RAN Distributed Unit
- **E2E** — end to end
- **ECR** — Elastic Container Registry
- **EFS** — Elastic File System
- **EKS** — Elastic Kubernetes Service
- **EPC** — Evolved Packet Core
- **IaC** — Infrastructure as Code
- **ISV** — Independent Software Vendor
- **MANO** — Management and Orchestration
- **MEC** — Multi-Access Edge Computing
- **NACL** — Network Access Control List
- **NAT** — Network Address Translation
- **NF** — Network Function
- **NFV** — Network Function Virtualization
- **NFVO** — Network Function Virtualization Orchestrator

- **NOC** — Network Operations Centre
- **NRF** — Network Repository Function
- **OSS** — Operations Support System
- **PII** — Personally Identifiable Information
- **QoS** — Quality of Service
- **RAN** — Radio Access Network
- **SBI** — Service Based Interface
- **SMF** — Session Management Function
- **SSL** — Secure Sockets Layer
- **TaS** — Test as Service
- **TCP** — Transmission Control Protocol
- **TLS** — Transport Layer Security
- **UDM** — Unified Data Management
- **UDP** — User Datagram Protocol
- **UPF** — User Plane Function
- **VIM** — Virtualized Infrastructure Manager
- **VNF** — Virtual Network Function
- **VPC** — Virtual Private Cloud

Notices

Customers are responsible for making their own independent assessment of the information in this document. This document: (a) is for informational purposes only, (b) represents current AWS product offerings and practices, which are subject to change without notice, and (c) does not create any commitments or assurances from AWS and its affiliates, suppliers or licensors. AWS products or services are provided “as is” without warranties, representations, or conditions of any kind, whether express or implied. The responsibilities and liabilities of AWS to its customers are controlled by AWS agreements, and this document is not part of, nor does it modify, any agreement between AWS and its customers.

© 2021 Amazon Web Services, Inc. or its affiliates. All rights reserved.