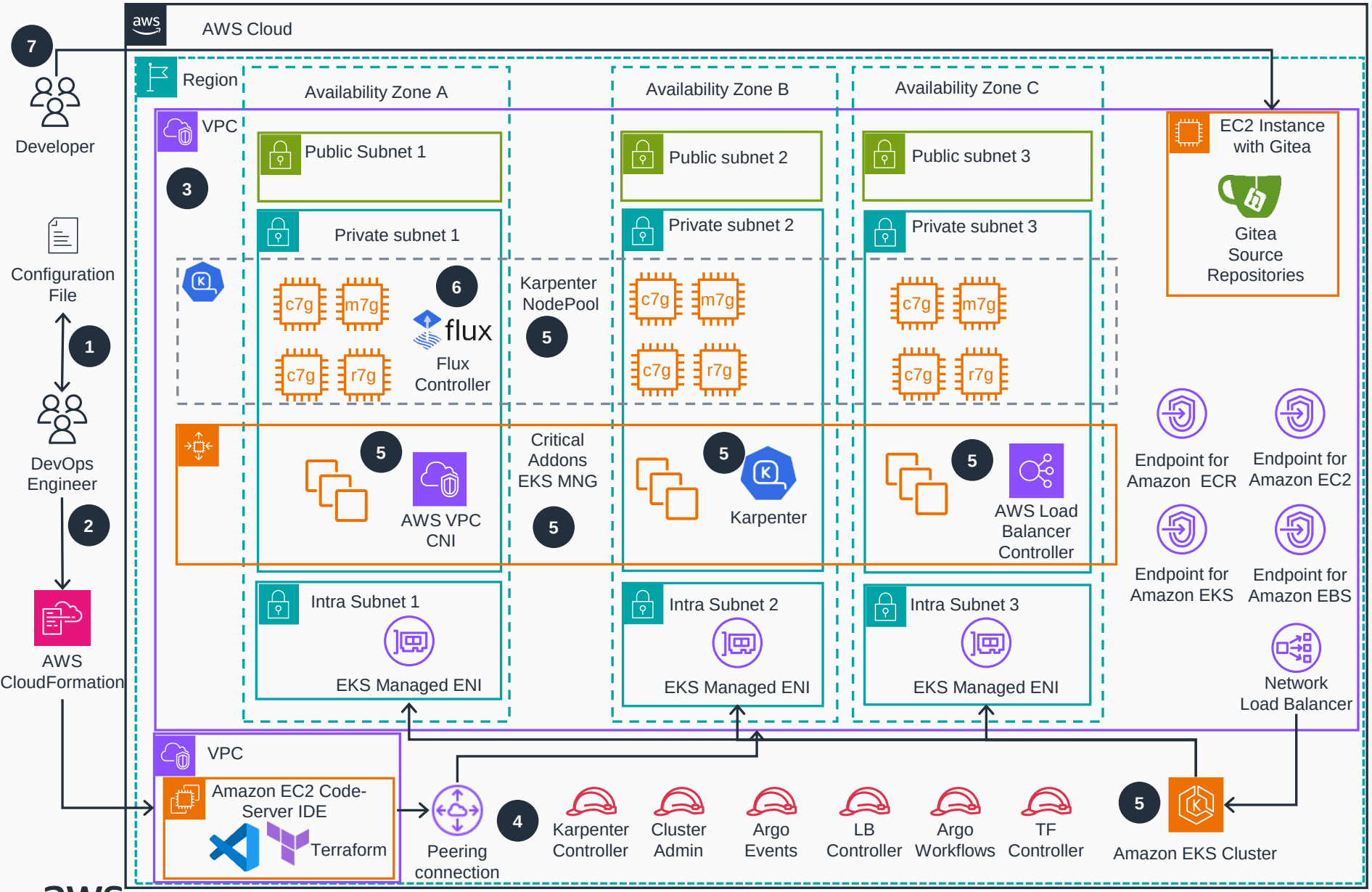
Guidance for Building SaaS applications on Amazon EKS using GitOps

This reference architecture shows how to provision an Amazon EKS cluster with critical add-ons.



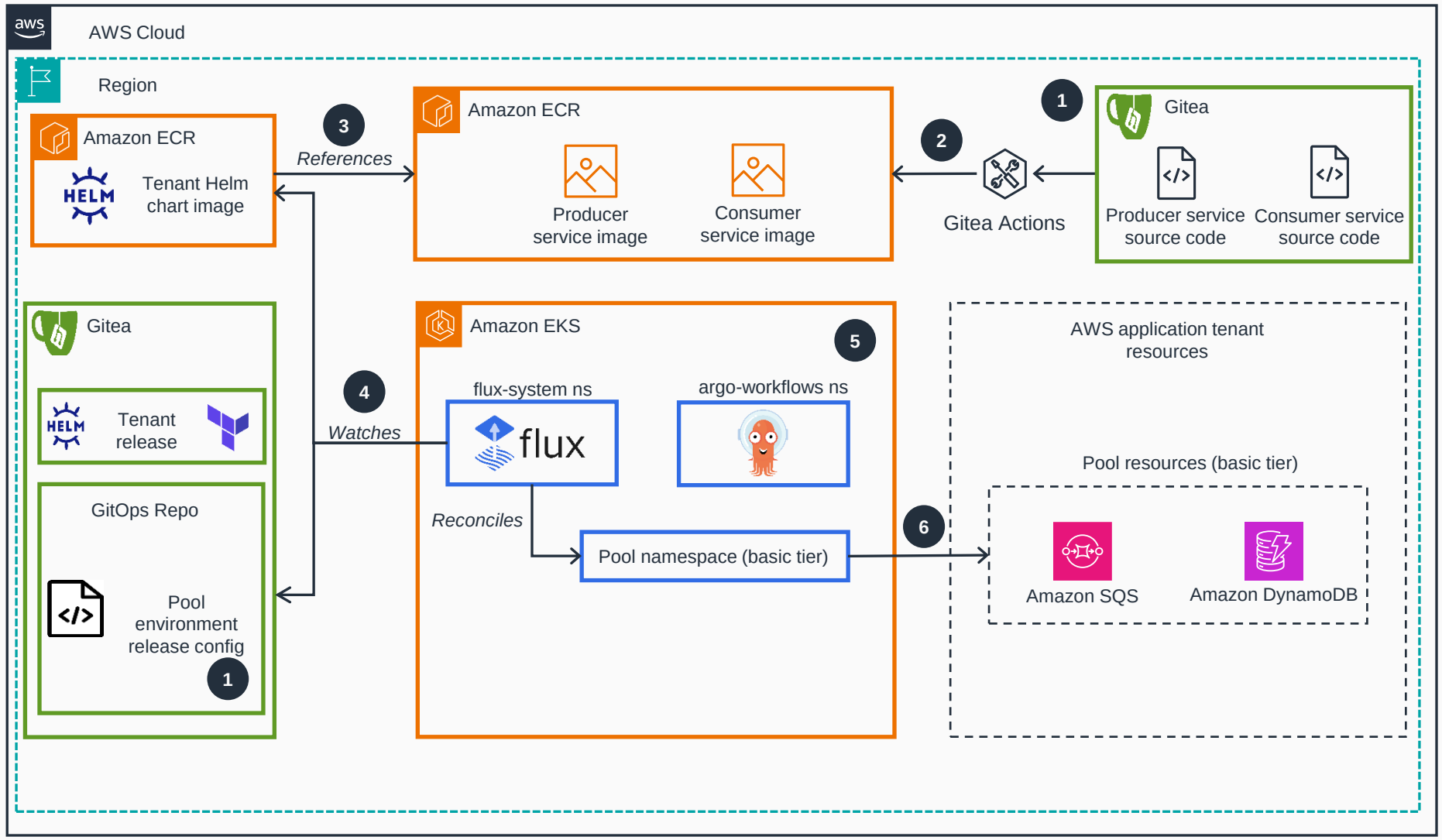
1. DevOps engineer defines a per-environment Terraform variable file that controls environment-specific configuration. This configuration file is used in all steps of deployment process by various configurations to provision different **Amazon Elastic Kubernetes Service (Amazon EKS)** environments.
2. DevOps engineer applies the environment configuration using Amazon CloudFormation which deploys an Amazon Elastic Compute Cloud (Amazon EC2) Instance with a VSCode IDE used to apply Terraform.
3. An Amazon Virtual Private Cloud (VPC) is provisioned and configured based on specified configuration. According to best practices for Reliability, three Availability Zones (AZs) are configured with corresponding VPC endpoints to provide access to resources deployed in private VPC and other VPC connected by VPC Peering.
4. User facing AWS Identity and Access Management (IAM) roles (Cluster Admin, Karpenter Controller, Argo Workflow, Argo Events, LB Controller, TF Controller) are created for various **Amazon EKS** cluster resources access levels, per Kubernetes security best practices.
5. Amazon EKS cluster is provisioned with Managed Nodes Group (MNG) that runs critical cluster add-ons (CoreDNS, AWS Load Balancer Controller, and Karpenter) on its compute node instances. Karpenter manages compute capacity to other add-ons, as well as business applications deployed by user while prioritizing provisioning Amazon EC2 Spot instances for the best price-performance.
6. Other important Amazon EKS add-ons (Flux controller etc.) are deployed based on the configurations defined in the per-environment Terraform configuration file (*see Step1 above*).
7. Gitea source code repositories running on Amazon EC2 can be accessed by Developer users to update microservices source code.

Reviewed for technical accuracy November 11, 2025

AWS Reference Architecture

Guidance for Building SaaS applications on Amazon EKS using GitOps

This architecture diagram shows a GitOps driven workflow on Amazon EKS clusters using FluxV2 for provisioning tenant resources.



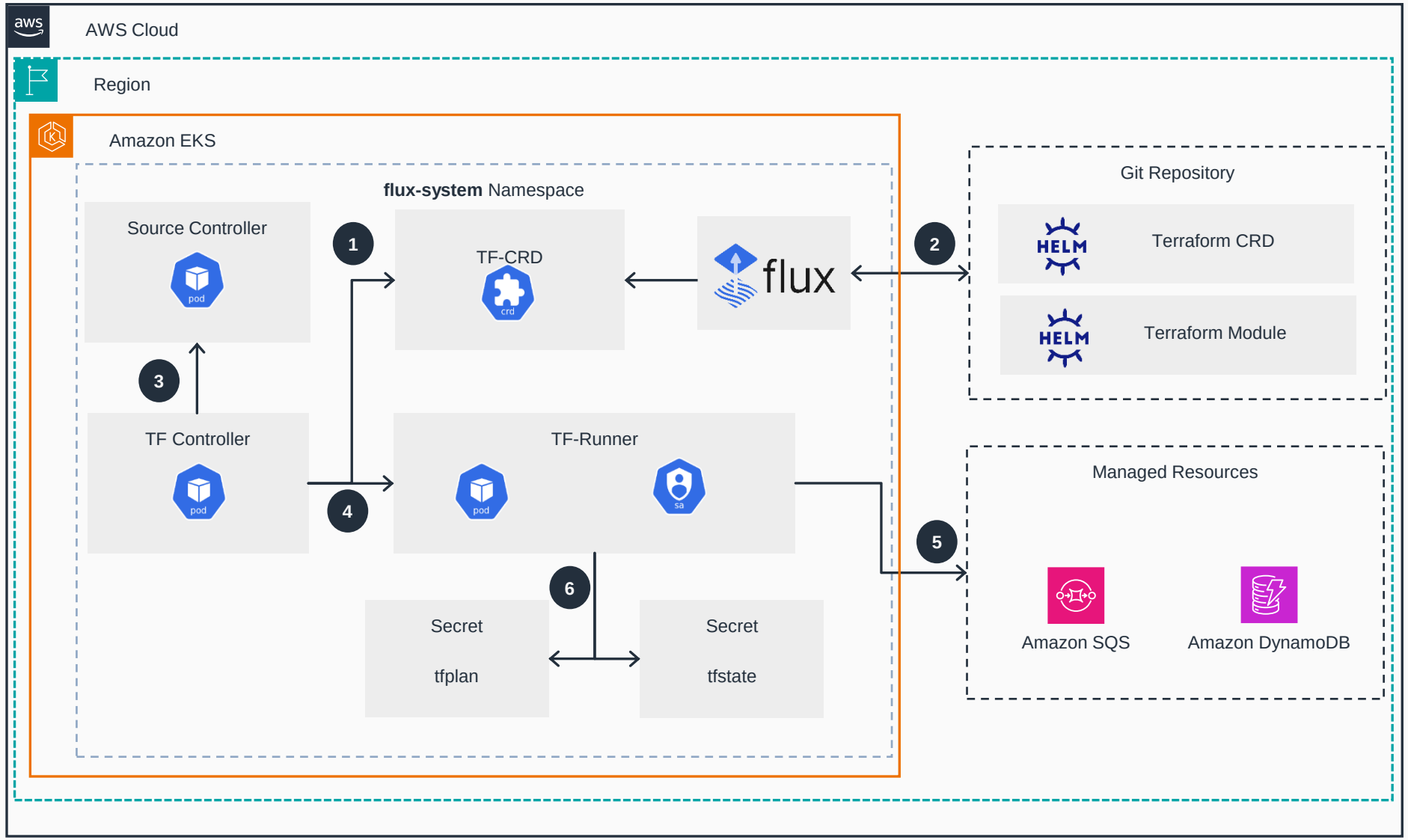
1. Gitea source code repositories hold the producer and consumer microservices application code, along with GitOps releases and Tenant resource definitions.
2. Gitea Actions is responsible to build the Producer and Consumer container images and push them to **Amazon Elastic Container Registry (ECR).**
3. **Amazon ECR** stores the Tenant Template Helm chart that references the producer and consumer service images.
4. Flux watches environment definition in Git and **Amazon ECR** to deploy changes to the **Amazon Elastic Kubernetes Service (Amazon EKS)** cluster, so that the cluster deployments match the expected state declared in the Git source repo and the correct version of Helm chart is deployed in the cluster.
5. The Argo Workflows controller is used for templating and automating variable replacement during onboarding, offboarding, and deployment processes. Argo Workflows automates these steps by committing the changes to the Git repository, which then triggers the rest of the GitOps pipeline.
6. Basic tier application tenants share AWS resources (Amazon Simple Queue Service (Amazon SQS) and Amazon DynamoDB). Basic tier tenants are served by the same microservice instances and infrastructure resources. This approach optimizes resource usage and reduces costs by sharing the infrastructure among multiple tenants.

Reviewed for technical accuracy November 11, 2025

AWS Reference Architecture

Guidance for Building SaaS applications on Amazon EKS using GitOps

This architecture diagram shows how Tofu Controller works with FluxV2 on Amazon EKS cluster to provision AWS managed resources through Terraform.



1. Flux continuously watches Git repositories for changes. In this case, it monitors the repository containing the Terraform Custom Resource Definition (CRD) and the Terraform module.
2. When a Terraform CRD is created in the cluster (defined in the Git repository), Flux detects this new resource and starts the reconciliation process.
3. The TF Controller is responsible for monitoring the Terraform CRD within the flux-system namespace. When it detects a new or updated Terraform CRD, it initiates the necessary actions.
4. The TF Controller launches a tf-runner pod. This pod pulls the specified Terraform module from the Git repository and executes it, managing the infrastructure as defined in the CRD.
5. The tf-runner pod provisions the required resources, such as Amazon Simple Storage Services (Amazon SQS) queues and Amazon DynamoDB tables, based on the Terraform module's definitions.
6. The state and plan of the Terraform execution are stored as Kubernetes secrets (e.g., tfstate and tfplan). This ensures that the state is preserved and can be accessed by subsequent Terraform operations.

Reviewed for technical accuracy November 11, 2025

AWS Reference Architecture