QuantSpark Weather App

About the Project

What we are building(Weather for London via API)?

We will be building a simple weather app using Go and the OpenWeatherMap API.

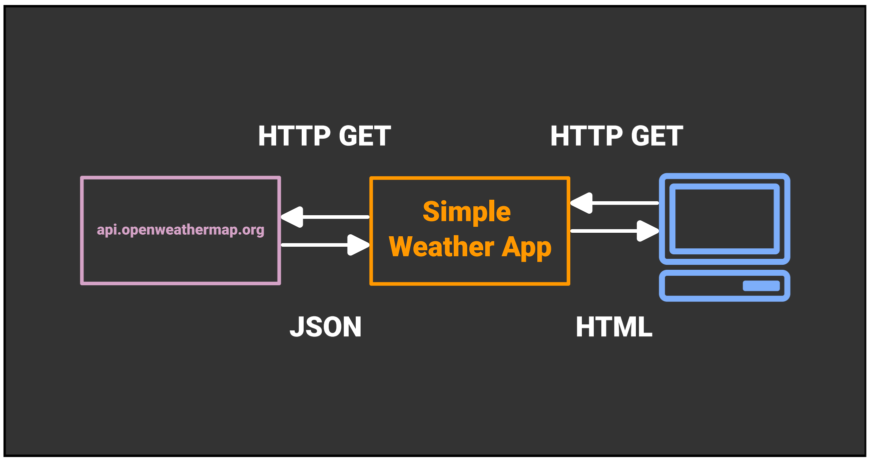


Image: App Architecture

Fetching Weather Data from OpenWeatherMap

Create an HTTP client using the net/http package.
Make a GET request to the OpenWeatherMap API to fetch weather data for a specific city.
Parse the JSON response using the encoding/json package.

Creating the HTML Template

Define the HTML structure for the weather app.
Use placeholders for the weather data that will be dynamically populated from the Go code.

Printing App Version and Container ID

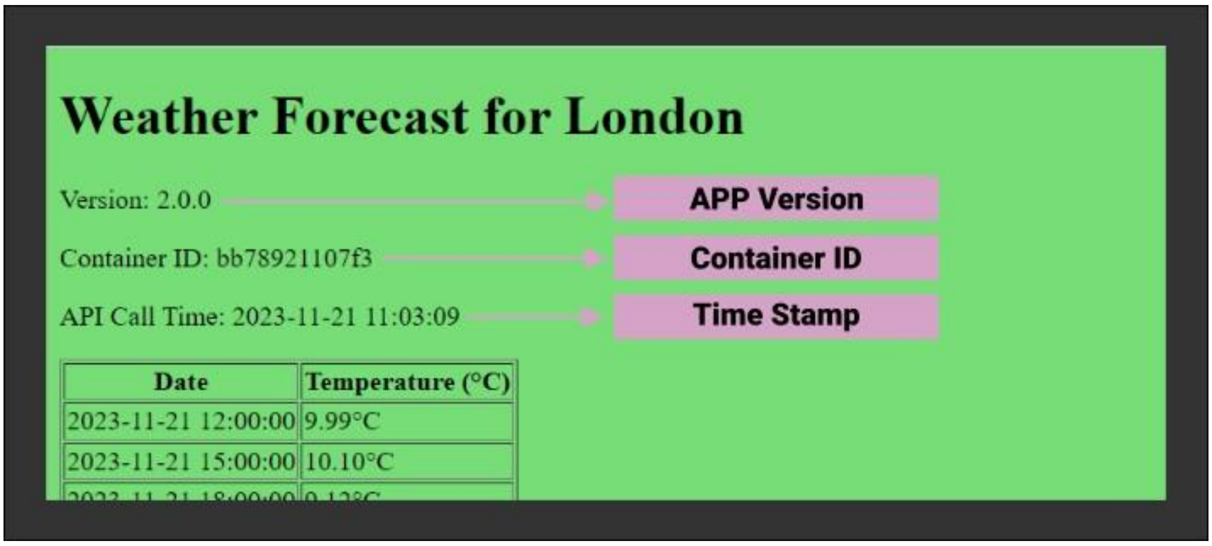
To provide additional information about the running instance of the weather app,
we can include the app version and container ID in the output of the application.
This can be useful for troubleshooting and debugging purposes.

Building the Web App

In the main.go file, write a Go program that fetches weather data from OpenWeatherMap, populates the HTML template with the data, and starts an HTTP server to serve the HTML page. Run the Go program using the go run command. Open your web browser and navigate to http://localhost:8080 to view the weather app.

App works as expected.

It's important to thoroughly test and validate our application before deploying it to production



Weather Forecast for London

Version: 2.0.0

Container ID: bb78921107f3

API Call Time: 2023-11-21 11:03:09

Date	Temperature (°C)
2023-11-21 12:00:00	9.99°C
2023-11-21 15:00:00	10.10°C

Image: App Version, Container ID and Date/Time Stamp

Build the Go App as a Binary

Then we build the Go app as a binary.
This will create an executable file that can be run on any system that has Go installed.
To build the binary, run the following command from the Go project directory:
go build -o main

Package the App and NGINX in a Container

Next, we will package the Go app binary and NGINX into a container.
This will allow us to run the app in a consistent and isolated environment, regardless of the underlying system. To create the container, we will use a Dockerfile.
A Dockerfile is a text file that contains instructions for building a Docker image.

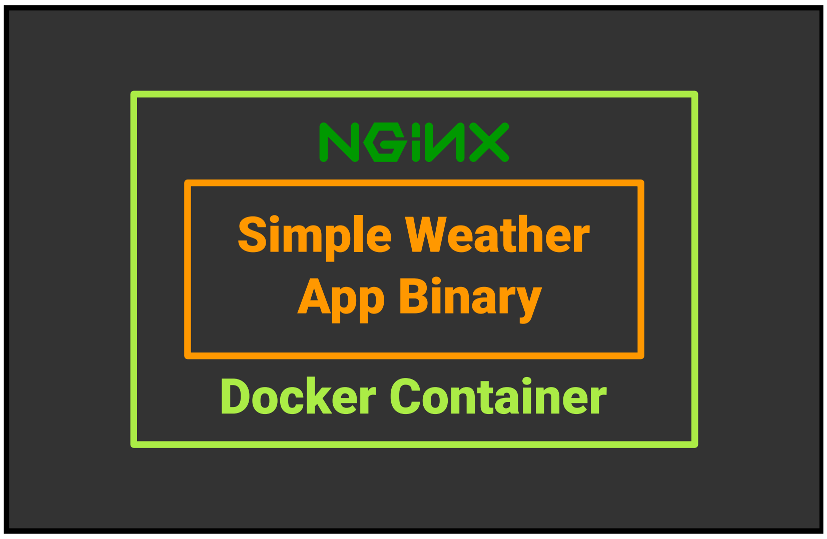
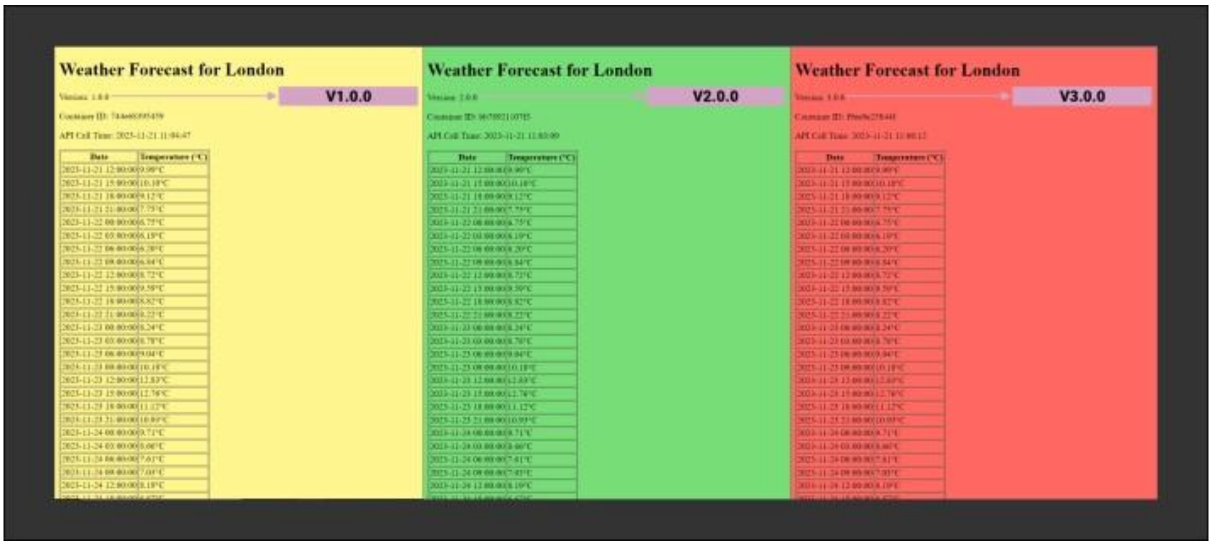


Image: App in Container

Run the Container

To run the Docker container, run the following command:
docker run -d -p 8080:8080 weather-app



Weather Forecast for London

Version: 1.0.0

Container ID: [illegible]

API Call Time: 2023-11-21 11:04:47

Date	Temperature (°C)
2023-11-21 12:00:00	9.99°C
2023-11-21 15:00:00	10.18°C
2023-11-21 18:00:00	9.12°C
2023-11-21 21:00:00	7.75°C
2023-11-22 00:00:00	6.75°C
2023-11-22 03:00:00	6.19°C
2023-11-22 06:00:00	6.20°C
2023-11-22 09:00:00	6.84°C
2023-11-22 12:00:00	8.72°C
2023-11-22 15:00:00	9.59°C
2023-11-22 18:00:00	8.82°C
2023-11-22 21:00:00	8.22°C
2023-11-23 00:00:00	8.24°C
2023-11-23 03:00:00	8.78°C
2023-11-23 06:00:00	9.04°C
2023-11-23 09:00:00	10.18°C
2023-11-23 12:00:00	12.83°C
2023-11-23 15:00:00	12.76°C
2023-11-23 18:00:00	11.12°C
2023-11-23 21:00:00	10.99°C
2023-11-24 00:00:00	9.71°C
2023-11-24 03:00:00	8.60°C
2023-11-24 06:00:00	7.61°C
2023-11-24 09:00:00	7.03°C
2023-11-24 12:00:00	8.19°C
[illegible]	[illegible]

Weather Forecast for London

Version: 2.0.0

Container ID: [illegible]

API Call Time: 2023-11-21 11:03:09

Date	Temperature (°C)
2023-11-21 12:00:00	9.99°C
2023-11-21 15:00:00	10.18°C
2023-11-21 18:00:00	9.12°C
2023-11-21 21:00:00	7.75°C
2023-11-22 00:00:00	6.75°C
2023-11-22 03:00:00	6.19°C
2023-11-22 06:00:00	6.20°C
2023-11-22 09:00:00	6.84°C
2023-11-22 12:00:00	8.72°C
2023-11-22 15:00:00	9.59°C
2023-11-22 18:00:00	8.82°C
2023-11-22 21:00:00	8.22°C
2023-11-23 00:00:00	8.24°C
2023-11-23 03:00:00	8.78°C
2023-11-23 06:00:00	9.04°C
2023-11-23 09:00:00	10.18°C
2023-11-23 12:00:00	12.83°C
2023-11-23 15:00:00	12.76°C
2023-11-23 18:00:00	11.12°C
2023-11-23 21:00:00	10.99°C
2023-11-24 00:00:00	9.71°C
2023-11-24 03:00:00	8.60°C
2023-11-24 06:00:00	7.61°C
2023-11-24 09:00:00	7.03°C
2023-11-24 12:00:00	8.19°C
[illegible]	[illegible]

Weather Forecast for London

Version: 3.0.0

Container ID: [illegible]

API Call Time: 2023-11-21 11:00:12

Date	Temperature (°C)
2023-11-21 12:00:00	9.99°C
2023-11-21 15:00:00	10.18°C
2023-11-21 18:00:00	9.12°C
2023-11-21 21:00:00	7.75°C
2023-11-22 00:00:00	6.75°C
2023-11-22 03:00:00	6.19°C
2023-11-22 06:00:00	6.20°C
2023-11-22 09:00:00	6.84°C
2023-11-22 12:00:00	8.72°C
2023-11-22 15:00:00	9.59°C
2023-11-22 18:00:00	8.82°C
2023-11-22 21:00:00	8.22°C
2023-11-23 00:00:00	8.24°C
2023-11-23 03:00:00	8.78°C
2023-11-23 06:00:00	9.04°C
2023-11-23 09:00:00	10.18°C
2023-11-23 12:00:00	12.83°C
2023-11-23 15:00:00	12.76°C
2023-11-23 18:00:00	11.12°C
2023-11-23 21:00:00	10.99°C
2023-11-24 00:00:00	9.71°C
2023-11-24 03:00:00	8.60°C
2023-11-24 06:00:00	7.61°C
2023-11-24 09:00:00	7.03°C
2023-11-24 12:00:00	8.19°C
[illegible]	[illegible]

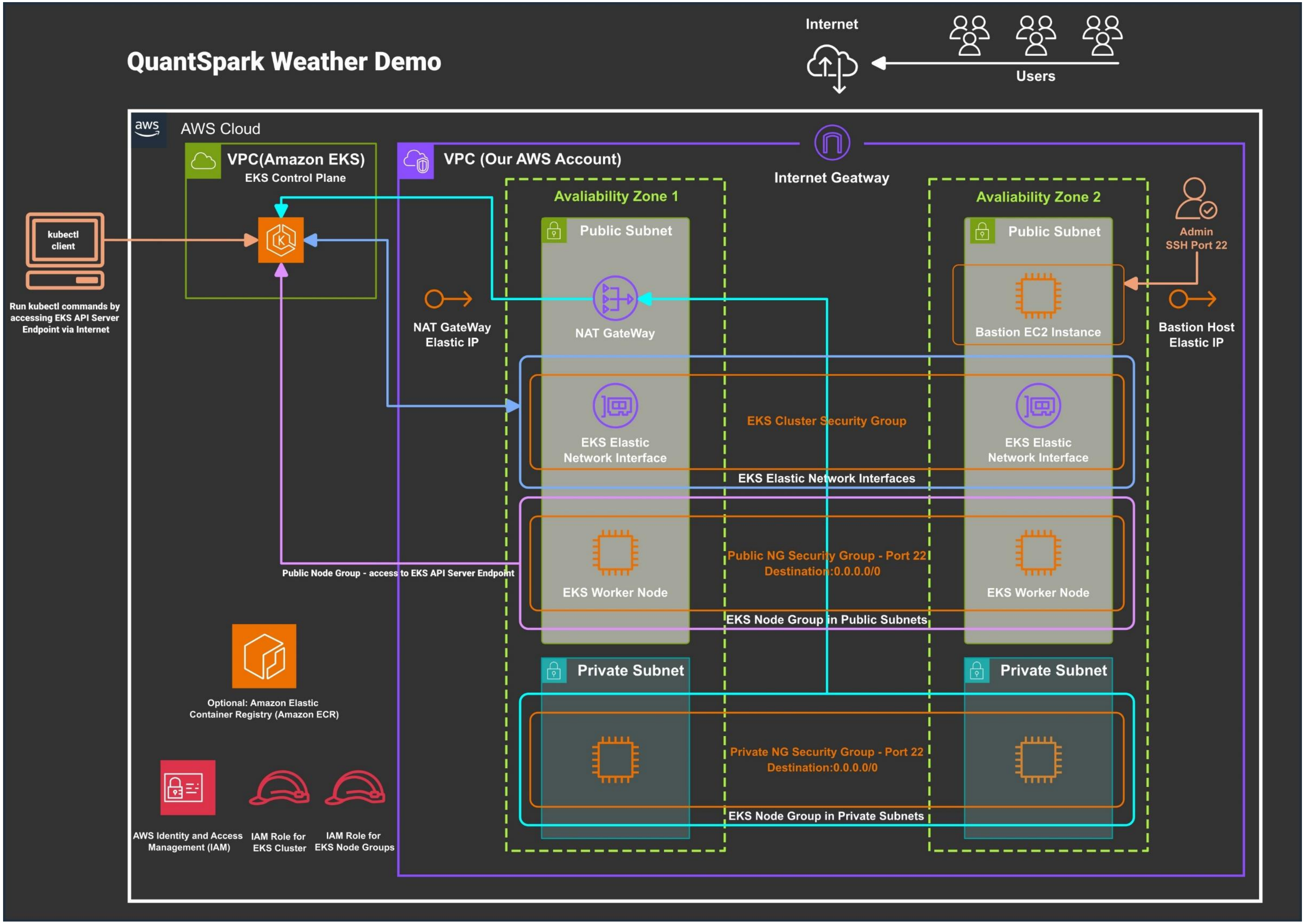
Image: App in three versions

Tag container and push to DockerHub

docker tag my-image:latest hatronix/weather:1.0.0
docker push hatronix/weather:1.0.0
docker tag my-image:latest hatronix/weather:2.0.0
docker push hatronix/weather:2.0.0
docker tag my-image:latest hatronix/weather:3.0.0
docker push hatronix/weather:3.0.0

Color-Coded Version Tags

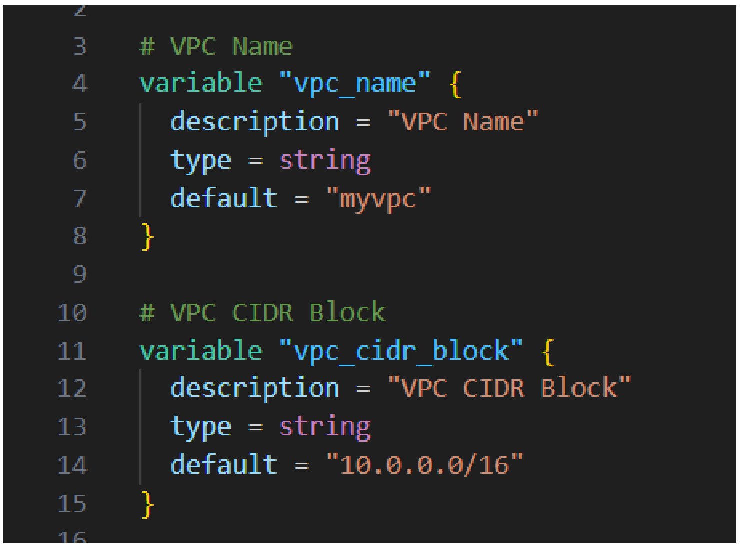
By tagging the containers with color-coded version tags (1.0.0 - yellow, 2.0.0 - green, 3.0.0 - red), it becomes easier to identify and track the deployed versions of the weather app. This visual distinction simplifies the process of upgrading or downgrading the deployment, allowing us to quickly recognize the specific image version you want to use.


QuantSpark Weather Demo
Internet
Users
AWS Cloud
VPC(Amazon EKS)
EKS Control Plane
VPC (Our AWS Account)
Internet Geatway
kubectl client
Run kubectl commands by accessing EKS API Server Endpoint via Internet
Avaliability Zone 1
Avaliability Zone 2
Public Subnet
Public Subnet
NAT GateWay Elastic IP
NAT GateWay
Bastion EC2 Instance
Admin SSH Port 22
Bastion Host Elastic IP
EKS Elastic Network Interface
EKS Cluster Security Group
EKS Elastic Network Interface
EKS Elastic Network Interfaces
EKS Worker Node
Public NG Security Group - Port 22 Destination:0.0.0.0/0
EKS Worker Node
EKS Node Group in Public Subnets
Public Node Group - access to EKS API Server Endpoint
Optional: Amazon Elastic Container Registry (Amazon ECR)
Private Subnet
Private Subnet
Private NG Security Group - Port 22 Destination:0.0.0.0/0
EKS Node Group in Private Subnets
AWS Identity and Access Management (IAM)
IAM Role for EKS Cluster
IAM Role for EKS Node Groups

Hosting Infrastructure (AWS)

What is VPC?

A virtual private cloud (VPC) is a logically isolated section of a public cloud infrastructure that provides a private network for computing resources.

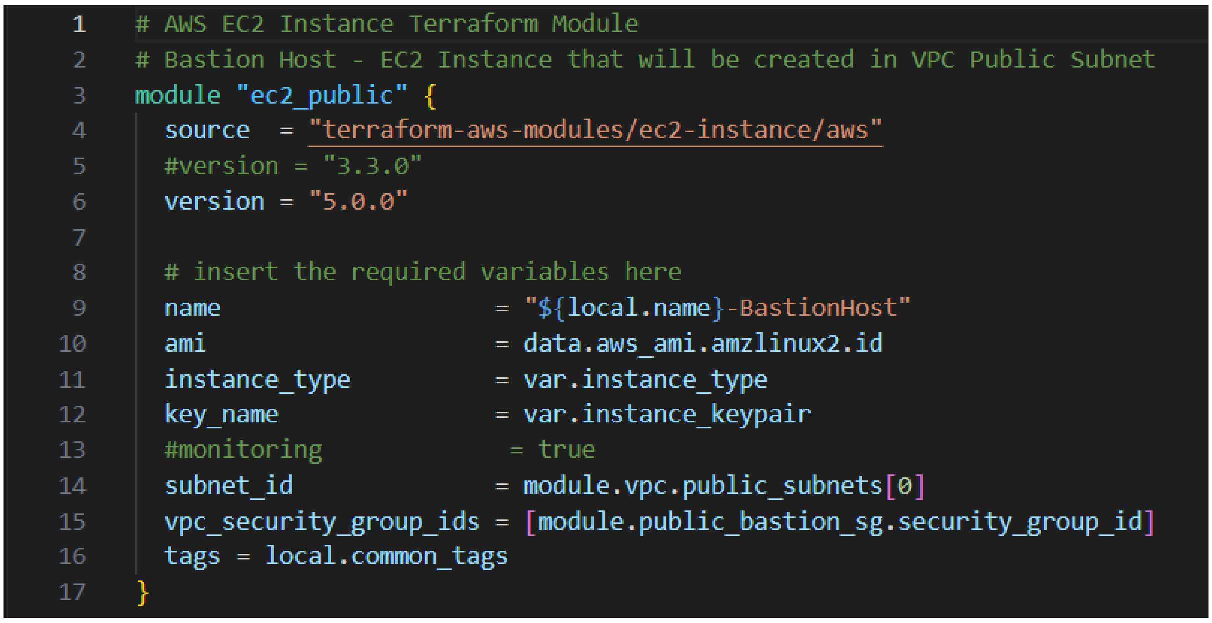


```
# VPC Name
variable "vpc_name" {
  description = "VPC Name"
  type = string
  default = "myvpc"
}

# VPC CIDR Block
variable "vpc_cidr_block" {
  description = "VPC CIDR Block"
  type = string
  default = "10.0.0.0/16"
}
```

Image: VPC

What is Bastion Host and why we need one?

A bastion host is a server that serves as an intermediary between an external network, such as the internet, and a private network, such as a virtual private cloud (VPC).
It acts as a secure gateway for authorized users to access resources within the private network. By centralizing access through the bastion host, we can enhance security by limiting direct exposure of private network resources to the public internet.

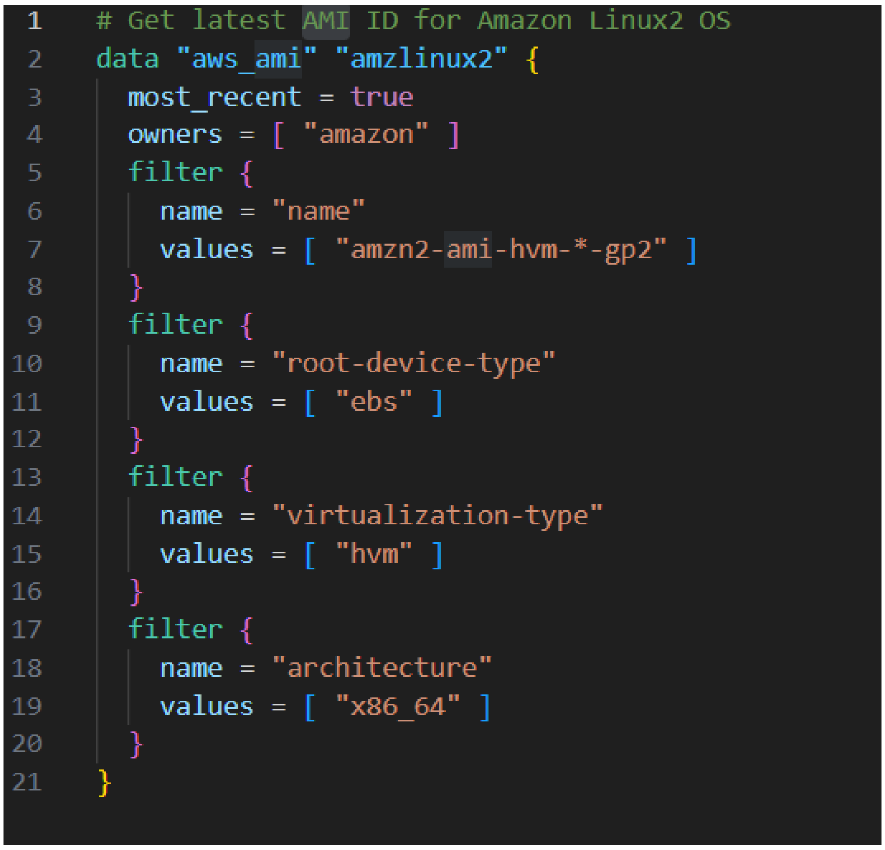


```
# AWS EC2 Instance Terraform Module
# Bastion Host - EC2 Instance that will be created in VPC Public Subnet
module "ec2_public" {
  source  = "terraform-aws-modules/ec2-instance/aws"
  #version = "3.3.0"
  version = "5.0.0"

  # insert the required variables here
  name                   = "${local.name}-BastionHost"
  ami                    = data.aws_ami.amzlinux2.id
  instance_type          = var.instance_type
  key_name               = var.instance_keypair
  #monitoring             = true
  subnet_id              = module.vpc.public_subnets[0]
  vpc_security_group_ids = [module.public_bastion_sg.security_group_id]
  tags = local.common_tags
}
```

Image: Bastion Host EC2 Instance

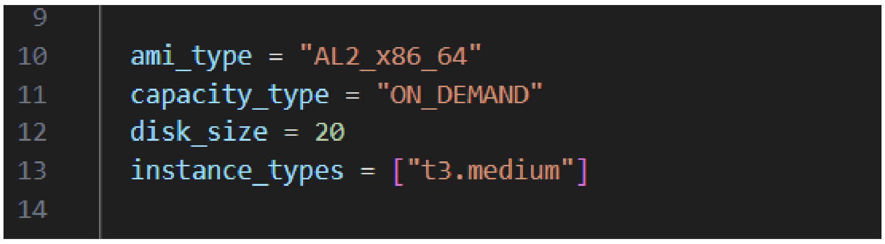
What is AMI image?

An Amazon Machine Image (AMI) is a pre-configured template that provides a starting point for launching Amazon Elastic Compute Cloud (EC2) instances.
AMIs contain software configurations, including an operating system, applications, and configurations for specific use cases.



```
# Get latest AMI ID for Amazon Linux2 OS
data "aws_ami" "amzlinux2" {
  most_recent = true
  owners = [ "amazon" ]
  filter {
    name = "name"
    values = [ "amzn2-ami-hvm-*-gp2" ]
  }
  filter {
    name = "root-device-type"
    values = [ "ebs" ]
  }
  filter {
    name = "virtualization-type"
    values = [ "hvm" ]
  }
  filter {
    name = "architecture"
    values = [ "x86_64" ]
  }
}
```

Image: Bastion Host AMI

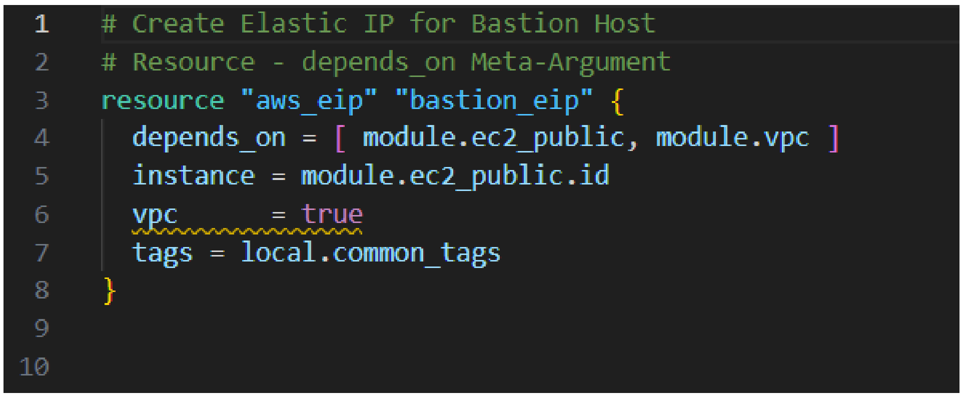


```

  ami_type = "AL2_x86_64"
  capacity_type = "ON_DEMAND"
  disk_size = 20
  instance_types = ["t3.medium"]

```

Image: AMI for EKS Public and Private Node Groups

What is ElasticIP?

An Elastic IP address is a public IP address that you can allocate to an Amazon Elastic Compute Cloud (EC2) instance in a specific region for dynamic cloud computing. It can be associated with an instance by updating the network interface attached to the instance.



```
# Create Elastic IP for Bastion Host
# Resource - depends_on Meta-Argument
resource "aws_eip" "bastion_eip" {
  depends_on = [ module.ec2_public, module.vpc ]
  instance = module.ec2_public.id
  vpc      = true
  tags = local.common_tags
}
```

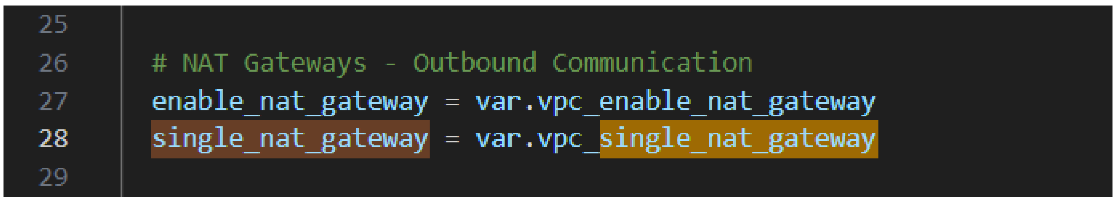
Image: ElasticIp for Bastion Host

How are Worker Nodes in Public Subnet getting outside data (Internet Getaway Scenario)?

In a standard cloud architecture, worker nodes in a public subnet typically access the internet through an internet gateway. The internet gateway is a VPC component that routes traffic between the VPC and the internet. Worker nodes can initiate outbound connections to the internet through the internet gateway, allowing them to retrieve data from external sources, such as public APIs, content delivery networks, and remote repositories.

How are Worker Nodes in Public Subnet getting outside data(Internet Getaway Scenario)?

In a standard cloud architecture, worker nodes in a public subnet typically access the internet through an internet gateway. The internet gateway is a VPC component that routes traffic between the VPC and the internet. Worker nodes can initiate outbound connections to the internet through the internet gateway, allowing them to retrieve data from external sources, such as public APIs, content delivery networks, and remote repositories.



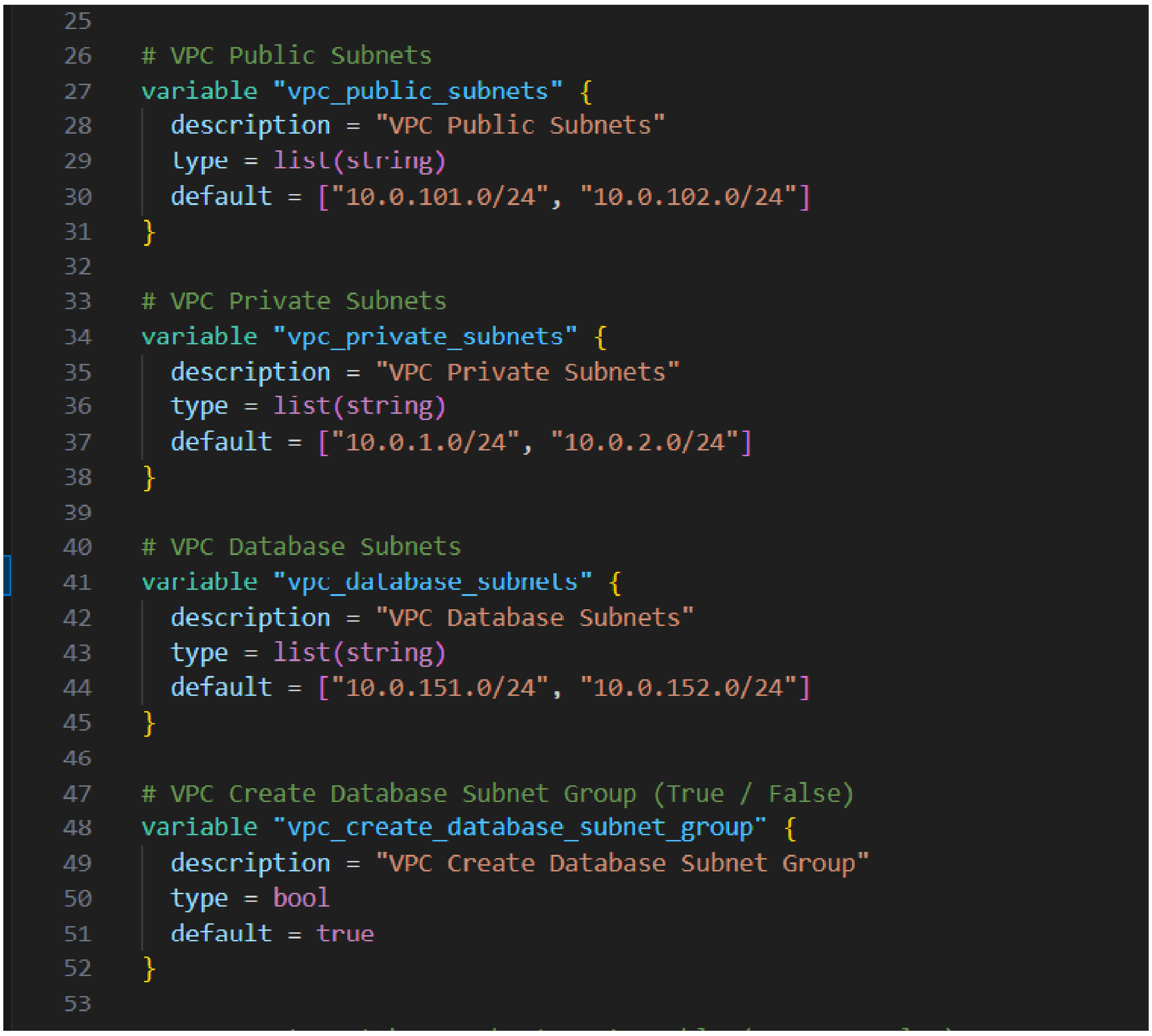
```

# NAT Gateways - Outbound Communication
enable_nat_gateway = var.vpc_enable_nat_gateway
single_nat_gateway = var.vpc_single_nat_gateway

```

Image: Enable NAT GateWay

What is Private Subnet?

In a virtual private cloud (VPC), a private subnet is a group of IP addresses within the VPC that are not directly accessible from the internet.
This means that resources in a private subnet cannot be directly reached by users or applications on the public internet.

Private subnets are often used to host sensitive resources, such as databases and application servers, to protect them from unauthorized access.



```
# VPC Public Subnets
variable "vpc_public_subnets" {
  description = "VPC Public Subnets"
  type = list(string)
  default = ["10.0.101.0/24", "10.0.102.0/24"]
}

# VPC Private Subnets
variable "vpc_private_subnets" {
  description = "VPC Private Subnets"
  type = list(string)
  default = ["10.0.1.0/24", "10.0.2.0/24"]
}

# VPC Database Subnets
variable "vpc_database_subnets" {
  description = "VPC Database Subnets"
  type = list(string)
  default = ["10.0.151.0/24", "10.0.152.0/24"]
}

# VPC Create Database Subnet Group (True / False)
variable "vpc_create_database_subnet_group" {
  description = "VPC Create Database Subnet Group"
  type = bool
  default = true
}
```

Image: VPC Subnets

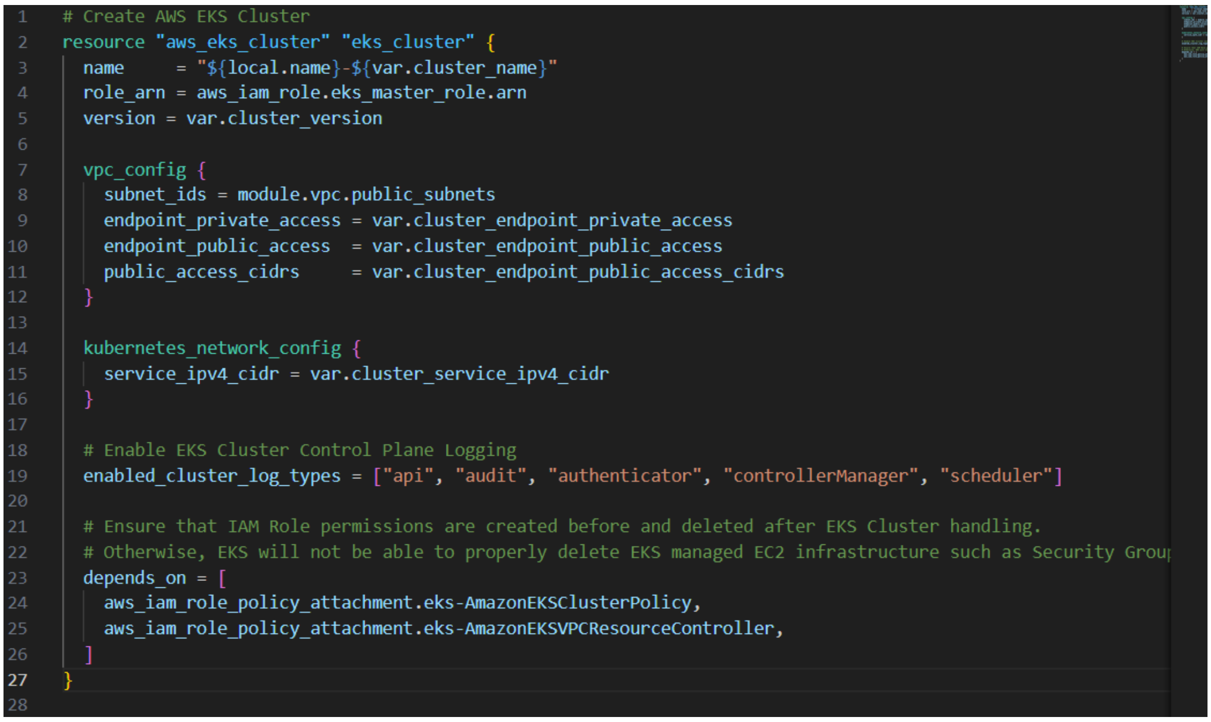
What is Public Subnet?

In a virtual private cloud (VPC), a public subnet is a group of IP addresses within the VPC that are directly accessible from the internet.
This means that resources in a public subnet can be directly reached by users or applications on the public internet.
Public subnets are often used to host web servers, load balancers, and other resources that need to be accessible from the internet.

What is AWS EKS Cluster?

An AWS EKS Cluster is a managed Kubernetes cluster that runs on Amazon Elastic Kubernetes Service (EKS). It provides a centralized control plane for managing your Kubernetes applications and infrastructure. EKS clusters are highly scalable and can be deployed in multiple availability zones to ensure high availability.
The enabled_cluster_log_types option specifies which types of logs to enable for your EKS cluster. The available log types are:

- api: Logs events related to the Kubernetes API server.
- audit: Logs events related to the Kubernetes audit system.
- authenticator: Logs events related to the Kubernetes authentication system.



```
# Create AWS EKS Cluster
resource "aws_eks_cluster" "eks_cluster" {
  name     = "${local.name}-${var.cluster_name}"
  role_arn = aws_iam_role.eks_master_role.arn
  version = var.cluster_version

  vpc_config {
    subnet_ids = module.vpc.public_subnets
    endpoint_private_access = var.cluster_endpoint_private_access
    endpoint_public_access  = var.cluster_endpoint_public_access
    public_access_cidrs     = var.cluster_endpoint_public_access_cidrs
  }

  kubernetes_network_config {
    service_ipv4_cidr = var.cluster_service_ipv4_cidr
  }

  # Enable EKS Cluster Control Plane Logging
  enabled_cluster_log_types = ["api", "audit", "authenticator", "controllerManager", "scheduler"]

  # Ensure that IAM Role permissions are created before and deleted after EKS Cluster handling.
  # Otherwise, EKS will not be able to properly delete EKS managed EC2 infrastructure such as Security Grou
  depends_on = [
    aws_iam_role_policy_attachment.eks-AmazonEKSClusterPolicy,
    aws_iam_role_policy_attachment.eks-AmazonEKSVPCResourceController,
  ]
}
```

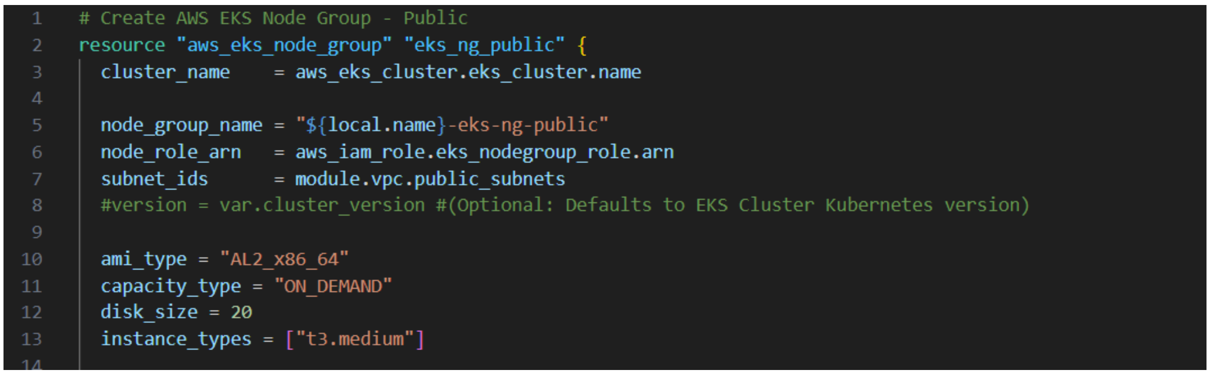
Image: EKS Cluster

What are EKS Elastic Interfaces?

EKS Elastic Interfaces (EFIs) are a networking feature of Amazon Elastic Kubernetes Service (EKS) that provides high-performance, low-latency network connectivity for containerized applications. EFIs enable direct communication between containerized applications on different EC2 instances, bypassing the need for traditional network interfaces and NAT gateways.

What is EKS Public Node Group?

An EKS Public Node Group is a type of managed node group in Amazon Elastic Kubernetes Service (EKS) that deploys EC2 instances into public subnets within the VPC. This allows the nodes to be directly accessible from the internet, enabling them to host applications that require public exposure, such as web servers, load balancers, and APIs.

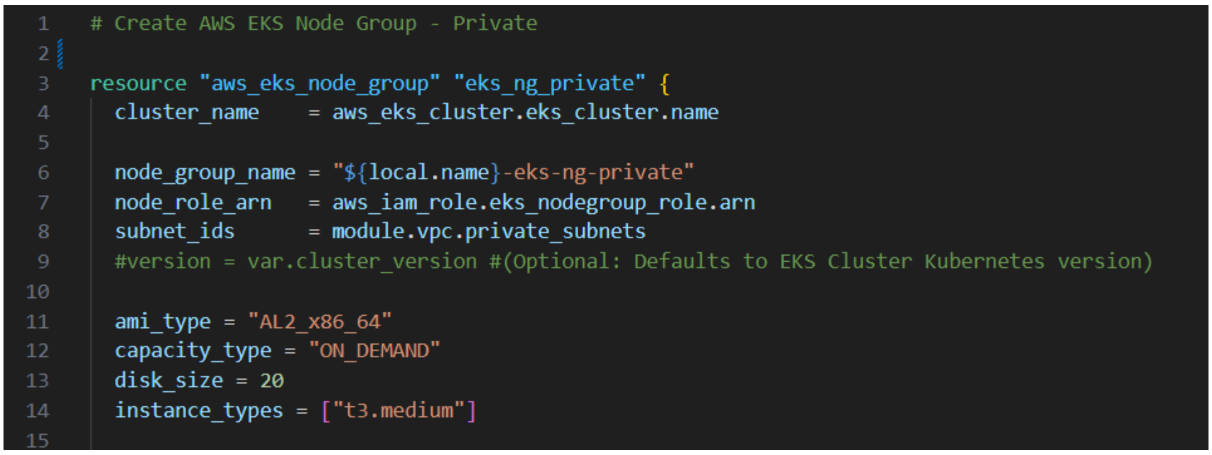


```
# Create AWS EKS Node Group - Public
resource "aws_eks_node_group" "eks_ng_public" {
  cluster_name    = aws_eks_cluster.eks_cluster.name

  node_group_name = "${local.name}-eks-ng-public"
  node_role_arn   = aws_iam_role.eks_nodegroup_role.arn
  subnet_ids      = module.vpc.public_subnets
  #version = var.cluster_version #(Optional: Defaults to EKS Cluster Kubernetes version)

  ami_type = "AL2_x86_64"
  capacity_type = "ON_DEMAND"
  disk_size = 20
  instance_types = ["t3.medium"]
```

Image: EKS Node Group Public

What is EKS Private Node Group?

An EKS Private Node Group is a type of managed node group in Amazon Elastic Kubernetes Service (EKS) that deploys EC2 instances into private subnets within the VPC. This prevents the nodes from being directly accessible from the internet, making them suitable for hosting applications that require isolation or security restrictions.



```
# Create AWS EKS Node Group - Private

resource "aws_eks_node_group" "eks_ng_private" {
  cluster_name    = aws_eks_cluster.eks_cluster.name

  node_group_name = "${local.name}-eks-ng-private"
  node_role_arn   = aws_iam_role.eks_nodegroup_role.arn
  subnet_ids      = module.vpc.private_subnets
  #version = var.cluster_version #(Optional: Defaults to EKS Cluster Kubernetes version)

  ami_type = "AL2_x86_64"
  capacity_type = "ON_DEMAND"
  disk_size = 20
  instance_types = ["t3.medium"]

```

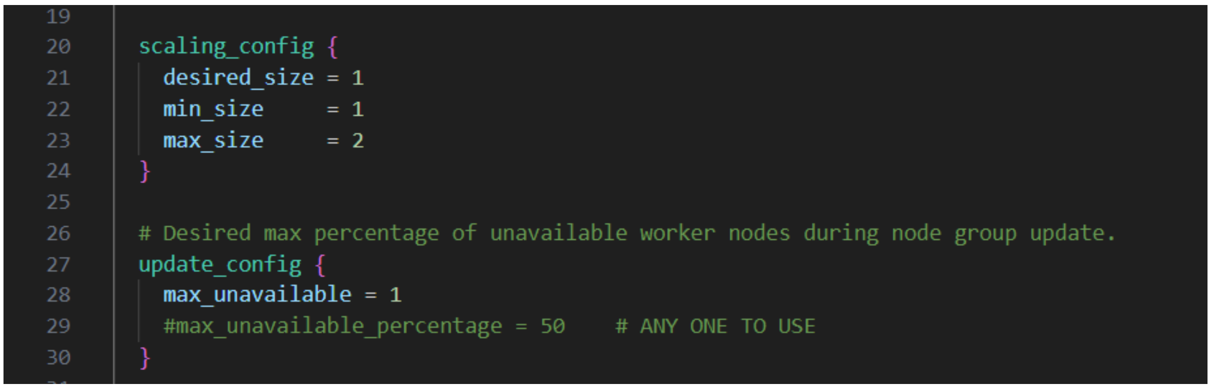
Image: EKS Node Group Public

Optional: What is Amazon Elastic Container registry?

Amazon Elastic Container Registry (ECR) is a fully managed container registry that makes it easy to store, manage, and deploy Docker and Open Container Initiative (OCI) images for your applications.

How can we scale EKS Node Groups horizontally?

Horizontally scaling EKS Node Groups involves adding or removing EC2 instances to an EKS cluster to adjust compute capacity based on changing workload demands. This process ensures that your applications have the resources they need to maintain performance and responsiveness, whether during periods of high traffic or low usage.

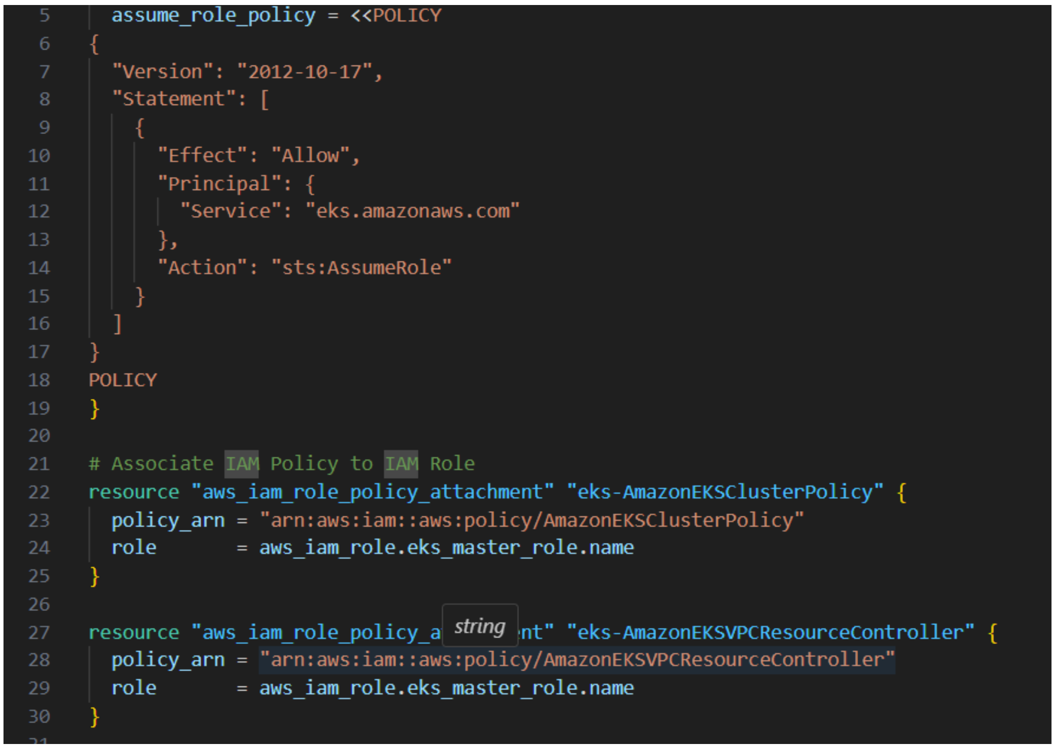


```
scaling_config {
  desired_size = 1
  min_size     = 1
  max_size     = 2
}

# Desired max percentage of unavailable worker nodes during node group update.
update_config {
  max_unavailable = 1
  #max_unavailable_percentage = 50    # ANY ONE TO USE
}
```

Image: EKS Node Group Scaling Configuration

What are IAM Role for EKS Cluster?

IAM Roles for EKS Clusters play a crucial role in securing and managing access to the Kubernetes control plane and nodes within an Amazon Elastic Kubernetes Service (EKS) cluster. They provide a mechanism for granting specific permissions to different entities, such as users, services, or applications, to perform authorized actions within the cluster.

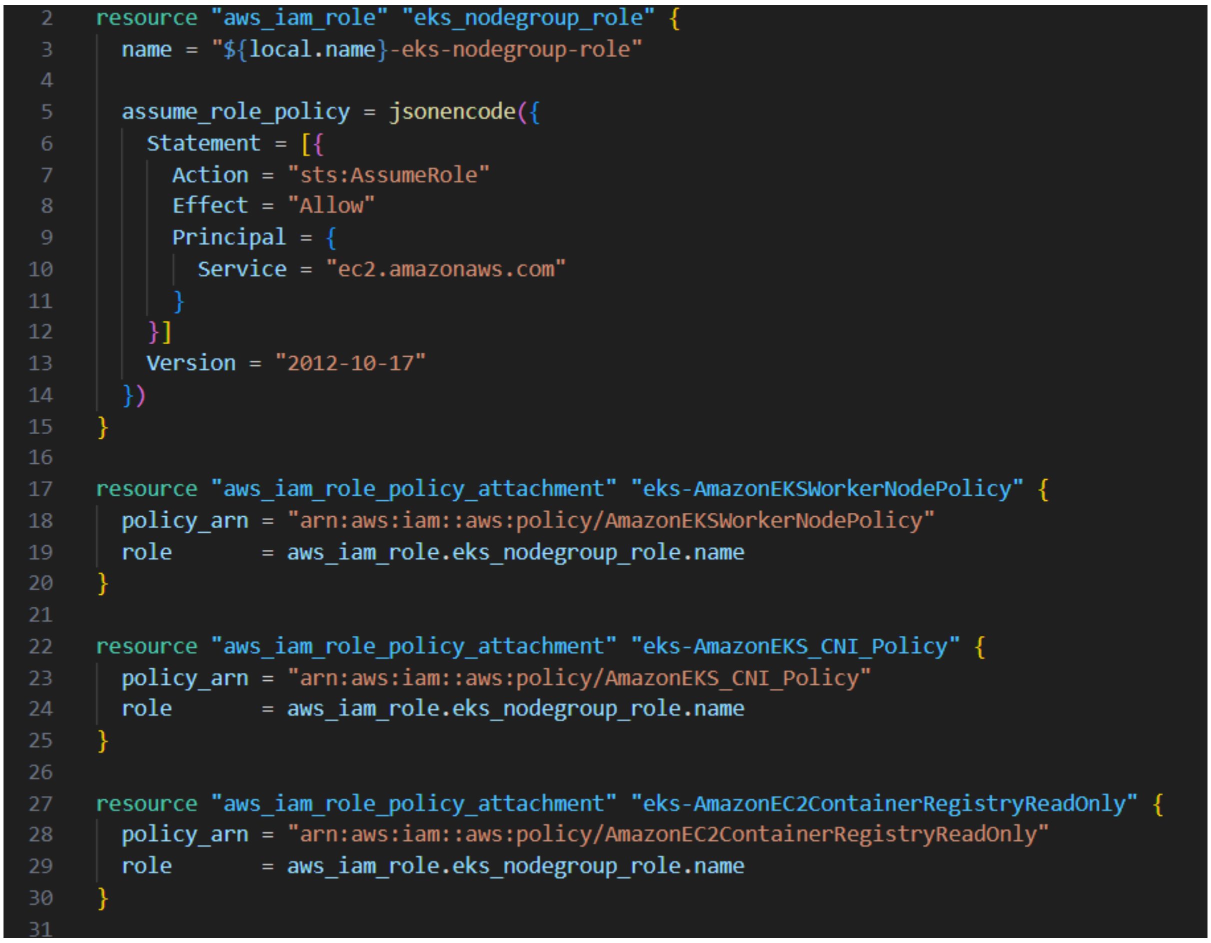


```
  assume_role_policy = <<POLICY
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "eks.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
POLICY
}

# Associate IAM Policy to IAM Role
resource "aws_iam_role_policy_attachment" "eks-AmazonEKSClusterPolicy" {
  policy_arn = "arn:aws:iam::aws:policy/AmazonEKSClusterPolicy"
  role       = aws_iam_role.eks_master_role.name
}

resource "aws_iam_role_policy_a string nt" "eks-AmazonEKSVPCResourceController" {
  policy_arn = "arn:aws:iam::aws:policy/AmazonEKSVPCResourceController"
  role       = aws_iam_role.eks_master_role.name
}
```

Image: IAM Role for EKS Cluster

What are IAM Role for EKS Node Group?

IAM Roles for EKS Node Groups play a crucial role in securely managing the EC2 instances that make up the worker nodes of an Amazon Elastic Kubernetes Service (EKS) cluster. They provide a mechanism for granting specific permissions to the EC2 instances, allowing them to perform authorized actions within the cluster and interact with AWS resources.



```
resource "aws_iam_role" "eks_nodegroup_role" {
  name = "${local.name}-eks-nodegroup-role"

  assume_role_policy = jsonencode({
    Statement = [{
      Action = "sts:AssumeRole"
      Effect = "Allow"
      Principal = {
        Service = "ec2.amazonaws.com"
      }
    }]
    Version = "2012-10-17"
  })
}

resource "aws_iam_role_policy_attachment" "eks-AmazonEKSWorkerNodePolicy" {
  policy_arn = "arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy"
  role       = aws_iam_role.eks_nodegroup_role.name
}

resource "aws_iam_role_policy_attachment" "eks-AmazonEKS_CNI_Policy" {
  policy_arn = "arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy"
  role       = aws_iam_role.eks_nodegroup_role.name
}

resource "aws_iam_role_policy_attachment" "eks-AmazonEC2ContainerRegistryReadOnly" {
  policy_arn = "arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly"
  role       = aws_iam_role.eks_nodegroup_role.name
}

```

Image: IAM Role for EKS Node Group

K8S and managing Deployment.

Kubernetes architecture

The Kubernetes architecture consists of two main components: AWS EKS Control Plane and EKS Managed Nodes in a Node Group

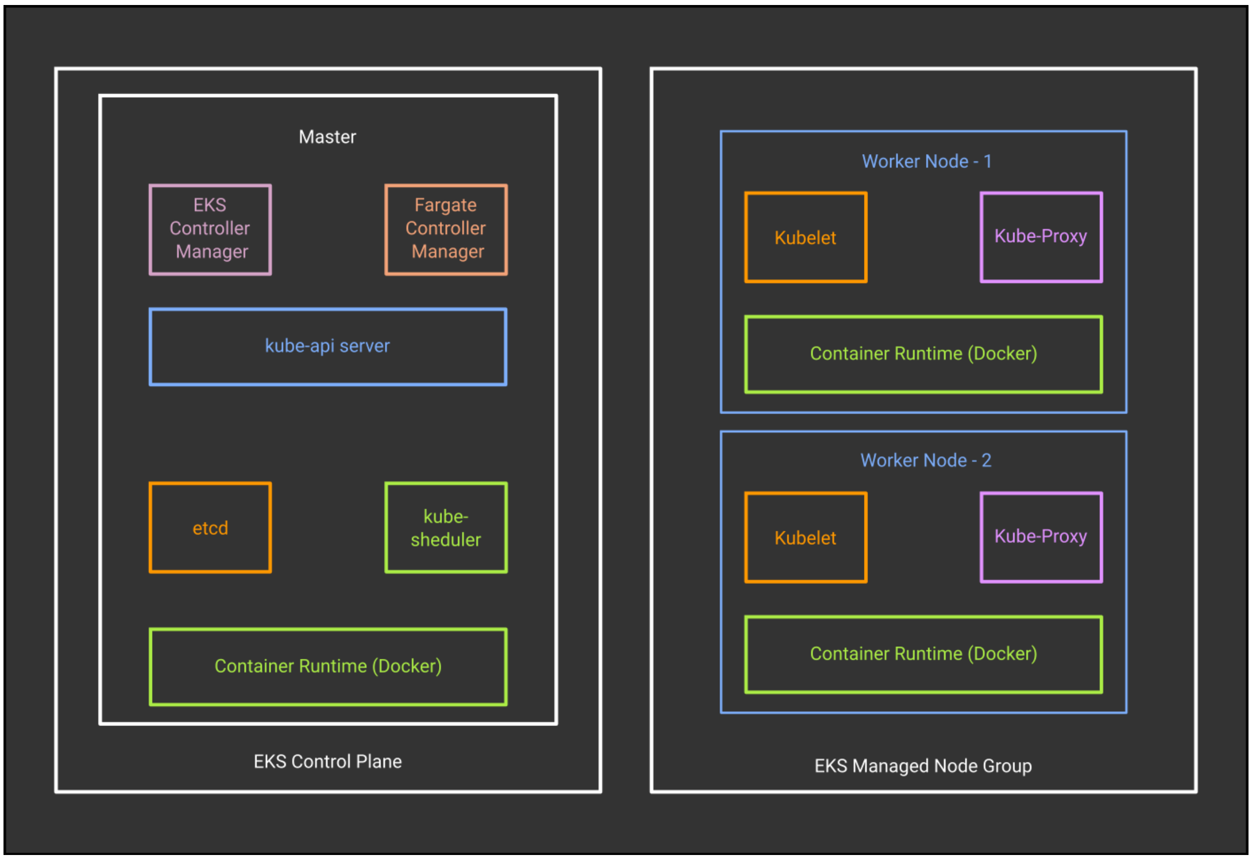


Image: K8S architecture

What is AWS EKS Control Plane?

The EKS control plane is the brains of your EKS cluster. It's responsible for managing the cluster's state, scheduling workloads, and making sure that your applications are running smoothly.

What is EKS Master node?

An EKS Master node, also known as a control plane node, is a specialized EC2 instance that manages the Kubernetes cluster in Amazon Elastic Kubernetes Service (EKS). It is responsible for running the Kubernetes control plane software, which includes the API server, kube-controller-manager, kube-scheduler, and etcd.

What is etcd in Master node?

Etcd is a distributed key-value store that serves as the backing store for all cluster data in Kubernetes. It is a highly available and replicated database that stores the cluster's configuration and state information.

What is kube-scheduler in Master node?

It is responsible for determining the most suitable node for each pod based on various factors, including resource availability, node affinity, and anti-affinity rules.

What is kube-api server in Master node?

Serves as the primary interface for interacting with the cluster. It acts as the front-end for the cluster, exposing a REST API that allows users and other components to manage cluster resources, such as pods, deployments, services, and namespaces.

What is EKS Controller Manager in Master node?

It is responsible for running several controllers that manage the lifecycle of cluster resources and maintain the overall state of the cluster.

- Node lifecycle management
- End-point slice management
- Service account management
- Cloud-specific control logic

What is Fargate controller manager?

Specifically handles the management of Fargate pods and their associated infrastructure. Fargate is a serverless compute engine for Kubernetes that allows users to run containers without provisioning and managing servers.
The Fargate Controller Manager is responsible for translating Kubernetes pod specifications into Fargate-specific configurations and managing the lifecycle of Fargate pods.

What is Kublet on a Worker Node?

The kubelet is an agent that runs on each worker node in a Kubernetes cluster. It is responsible for managing the pods that are scheduled to run on the node. The kubelet communicates with the control plane to get the latest pod specifications and then takes the necessary steps to ensure that the pods are running and healthy.

What is Kube-Proxy on a Worker Node?

Kube-proxy is a network proxy that runs on each worker node in a Kubernetes cluster. It is responsible for implementing the Kubernetes service abstraction, which allows pods to communicate with each other using service names and ports. Kube-proxy maintains network rules on nodes to allow network communication to pods from network sessions inside or outside of the cluster.

K8S Fundamentals

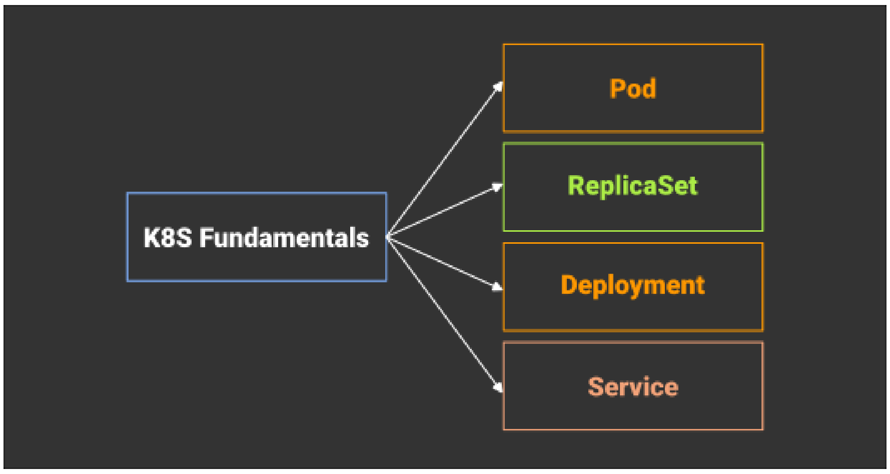


Image: K8S architecture

Pods: A pod is the basic unit of deployment in Kubernetes. It represents a group of one or more containers that are deployed together on a worker node and share resources.

Replica Set: Will maintain a stable set of replica Pods running at any given time. It is used to guarantee the availability of a specified number of identical Pods.

Deployments: A deployment manages the creation and updating of pods. It ensures that a specified number of pods are always running, even if some of them fail.

Services: A service abstracts away the IP addresses and port numbers of pods, providing a single, stable endpoint for accessing pods. It acts as a load balancer, distributing traffic across multiple pods.

Namespaces: Namespaces provide a way to logically isolate groups of resources, such as pods, services, and deployments, within a Kubernetes cluster. They help prevent conflicts between resources and simplify management.

How it is all working together in K8S?

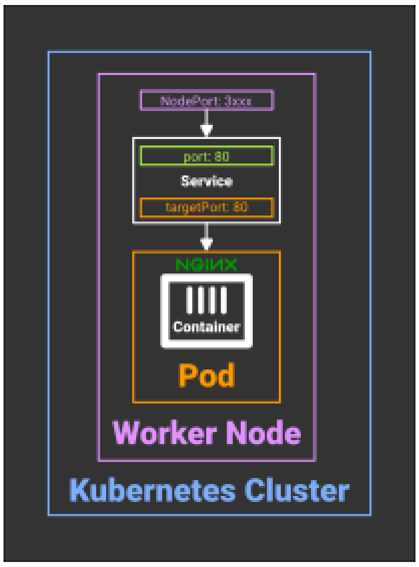


Image: K8S All working Together

Container: A lightweight and isolated environment that encapsulates an application's code, dependencies, and runtime. It allows applications to run consistently and reliably across different computing environments.

Pod: We can have a group of one or more containers that are tightly coupled and share resources. Pods are the basic unit of deployment in Kubernetes.

Worker Node: A physical machine that runs containers. Worker nodes provide the resources necessary for containers to execute.

Target Port: The port number inside a container where the application is listening for requests.

Port: The port number that is exposed to the outside world and that services use to communicate with the application.

Node Port: A port number that is opened on each worker node and that allows external traffic to reach the application's target port.

Kubernetes Cluster: A group of worker nodes managed by a control plane. Kubernetes clusters provide a platform for deploying, managing, and scaling containerized applications.

How are ReplicaSets working?

A ReplicaSet is a Kubernetes resource that ensures that a specified number of replica pods are running at any given time. It is used to manage the scaling and availability of pods in a Kubernetes cluster. ReplicaSets work by creating and deleting pods as needed to reach the desired number of replicas.

Let's Imagine that we have two identical pods in our deployment, each running the same container with the same app inside.

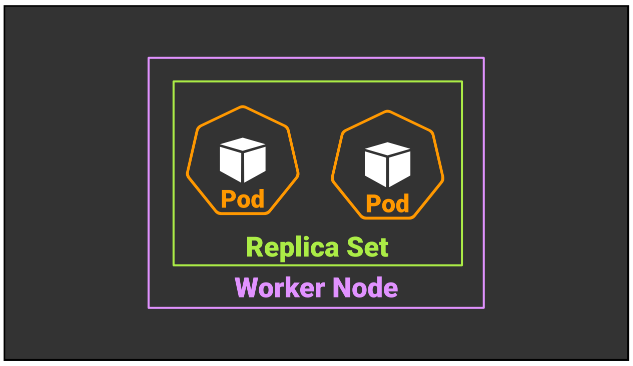


Image: Two identical pods belonging to a replica set

Now let's assume that for some reason, of the identical pods has stopped working.

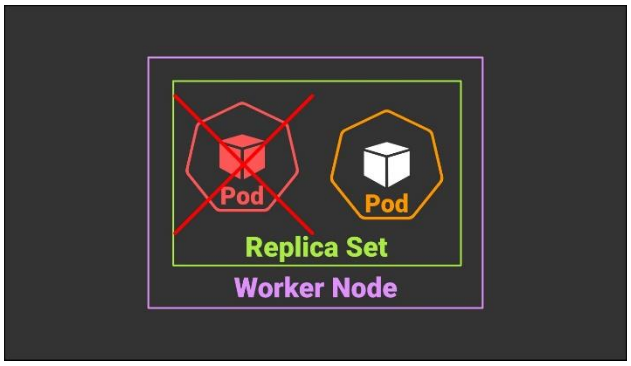


Image: One pods stops working

ReplicaSet will maintain a stable set of replica Pods running at any given time. It is used to guarantee the availability of a specified number of identical Pods. So the same pod will be recreated.

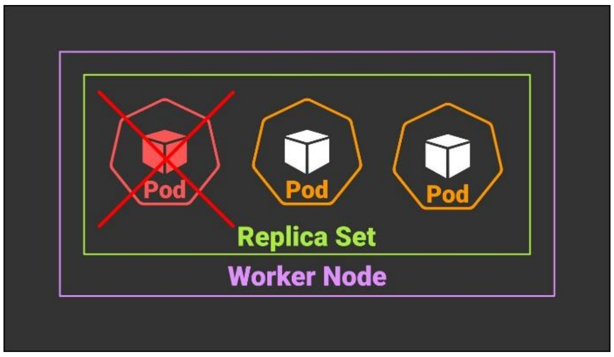


Image: The same pod is recreated by the replica set

Why we use replica sets?

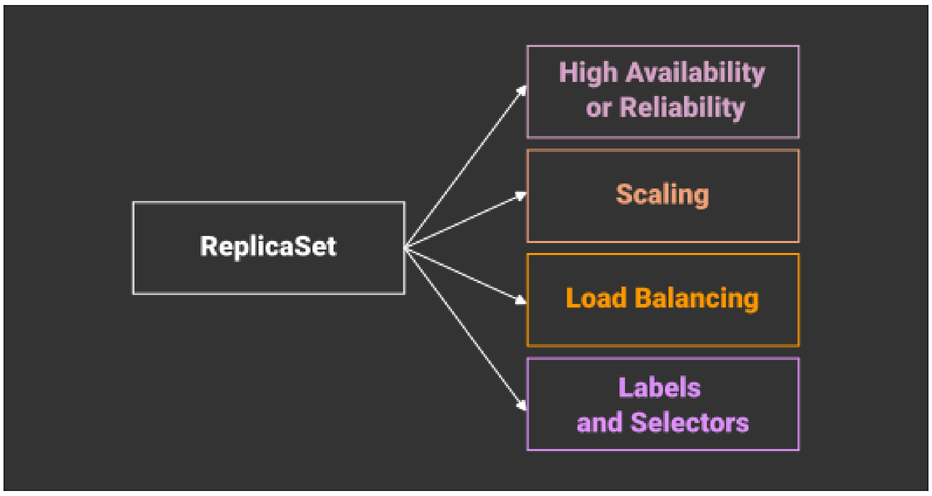


Image: Why we use Replica Sets

Replica sets are a fundamental component of Kubernetes for managing the deployment and scaling of containerized applications. They provide several benefits that make them essential for production environments:

Automated Pod Management: Replica sets automate the creation and deletion of pods, reducing the operational overhead of managing pods manually. This frees up developers and DevOps engineers to focus on more value-added tasks.

Scalability: Replica sets enable easy scaling of applications up or down based on demand. This ensures that applications can handle fluctuating workloads efficiently, providing resources when needed and reducing costs when demand is low.

High Availability: Replica sets maintain the desired number of replica pods, ensuring that applications are always available even if some pods fail or are terminated. This improves application resilience and reduces downtime.

Simplified Deployment: Replica sets simplify the deployment process by encapsulating the desired state of an application in a single object. This makes it easier to deploy and manage applications across multiple clusters and environments.

Health Monitoring: Replica sets continuously monitor the health of pods and automatically restart or replace unhealthy pods. This helps maintain application uptime and prevent service disruptions.

Resource Management: Replica sets consider resource constraints when scheduling pods onto worker nodes, ensuring that applications have the resources they need to run smoothly. This prevents resource contention and performance bottlenecks.

What are K8S Services?

Services act as an abstraction layer, decoupling application endpoints from their underlying pod IPs. They provide a stable and consistent way for other pods or external clients to communicate with the application, regardless of the pod IPs changing due to pod scheduling or restarts. Services also enable load balancing, distributing incoming traffic across multiple pods to improve performance and responsiveness.

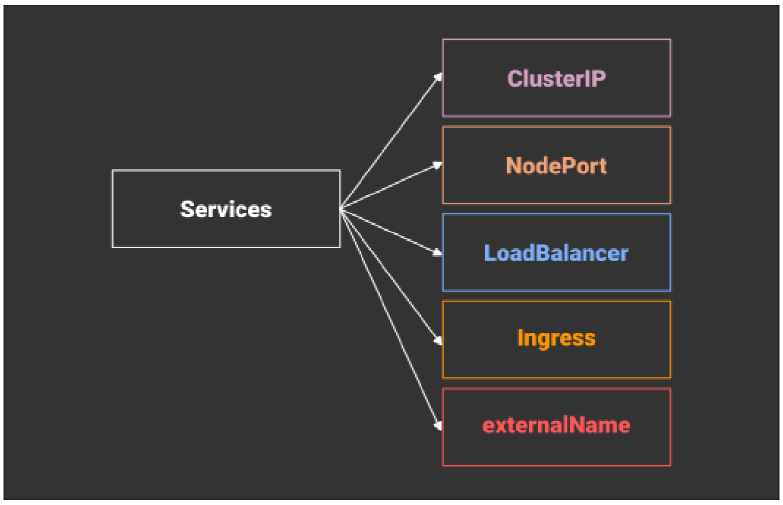


Image: Type of Services in K8S

Abstracting Pod Endpoints: ClusterIP
Facilitates communication between applications within the Kubernetes cluster.
Example: Frontend application communicating with a backend application.

Abstracting Pod Endpoints: NodePort
Enables external access to applications in the Kubernetes cluster using a Worker Node Port.
Example: Accessing a Frontend application through a web browser.

Abstracting Pod Endpoints: LoadBalancer
Integrates with cloud providers' Load Balancer services.
Example: Utilizing AWS Elastic Load Balancer for load balancing.

Abstracting Pod Endpoints: Ingress
An advanced load balancer offering features such as context path-based routing, SSL, SSL redirection, and more.
Primarily used with Layer 7 (the Application layer).
Example: Configuring AWS Application Load Balancer (ALB) for sophisticated load balancing.

Interplay of ReplicaSets and Services in K8S?

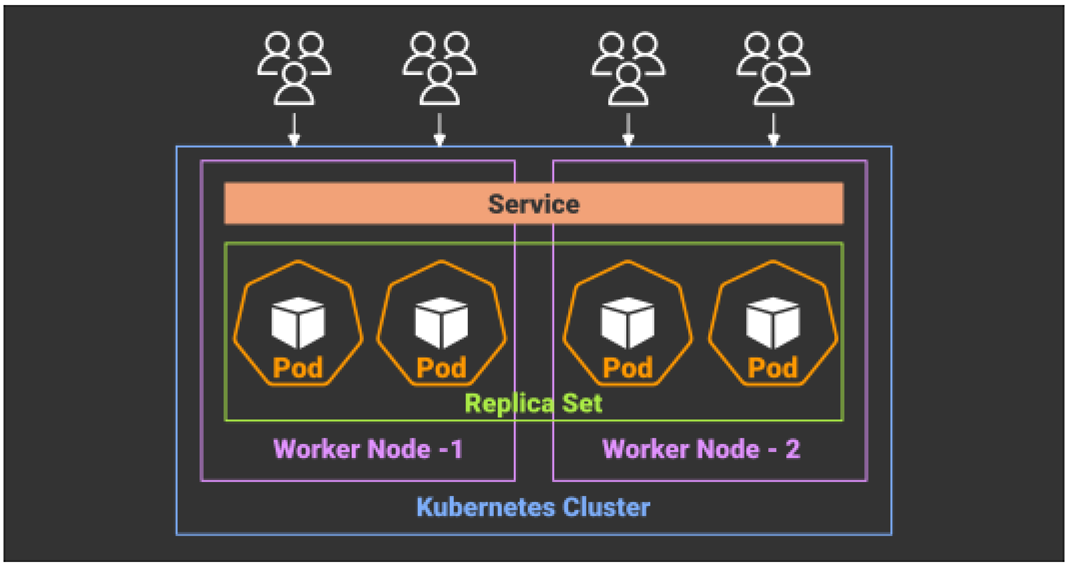


Image: Services and ReplicaSets in K8S

ReplicaSets maintain the desired number of replica pods, while Services provide a stable abstraction layer for accessing and load balancing traffic across those pods. Together, they form a powerful foundation for managing applications in a Kubernetes cluster.

What is K8S Deployment?

A deployment is a higher-level abstraction in Kubernetes that manages the creation and updating of pods. It is responsible for ensuring that the desired number of pods are always running, even if some pods fail or are terminated. Deployments also provide a declarative way to describe the desired state of an application, which the Kubernetes controller will automatically reconcile with the actual state.

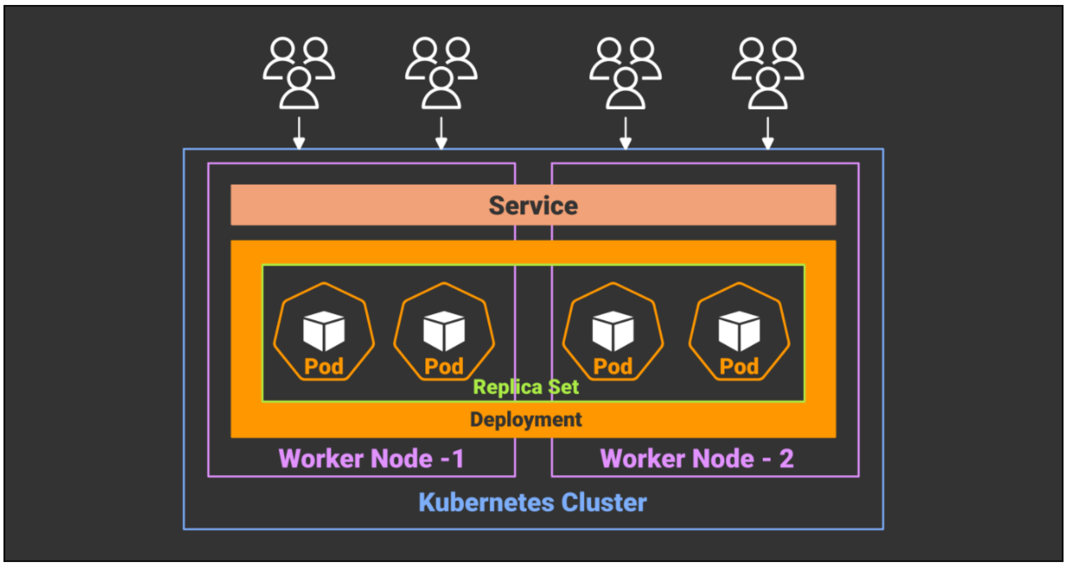


Image: K8S Deployment

Relationship between Deployments, ReplicaSets, and Pods?

Deployments manage ReplicaSets, which in turn manage pods. This creates a hierarchical relationship between the three resources:

- A deployment can manage multiple ReplicaSets.
- A ReplicaSet can manage multiple pods.
- A pod can only belong to one ReplicaSet.

When a deployment is created, it creates a ReplicaSet with the desired number of replicas. The ReplicaSet then creates the pods and monitors their health. If a pod fails or is terminated, the ReplicaSet will automatically create a new pod to replace it.