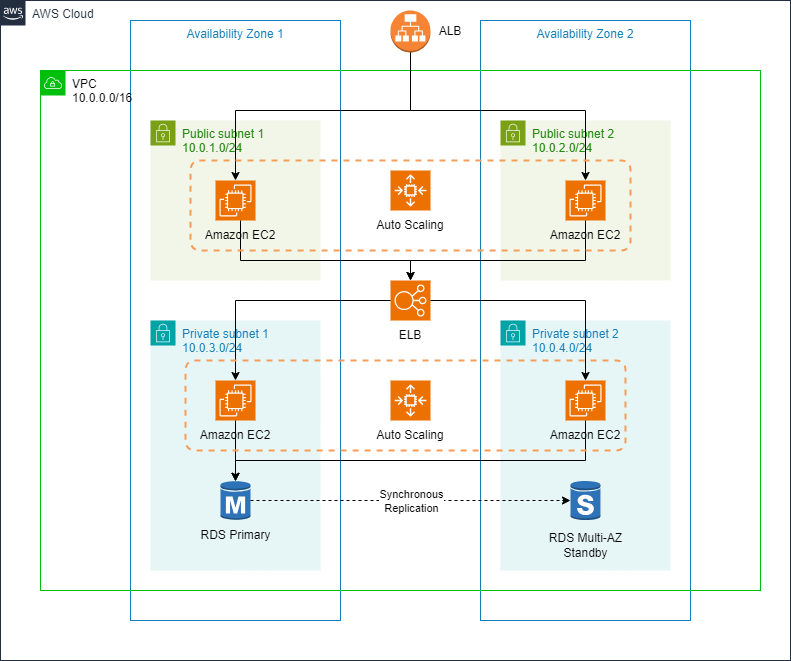
**AWS Architecture Documentation**

****

**1. AWS Services Used and Reasoning for Service Choices**

**1.1 Virtual Private Cloud (VPC)**

* **Service**: Amazon VPC provisions a logically isolated section of the AWS cloud where you can launch AWS resources in a virtual network.
* **Reasoning**: VPC provides complete control over the network configuration, including IP address ranges, subnets, route tables, and gateways. It isolates the infrastructure and enables secure connectivity between resources.

**1.2 Subnets**

* **Service**: Amazon VPC Subnets divide VPC into smaller networks.
* **Reasoning**: By creating public and private subnets, we can separate resources that need direct internet access from those that don't, improving security. Public-facing components (like the ALB) are placed in public subnets, while backend components (like ECS and RDS) are placed in private subnets.

**1.3 Auto Scaling Group (ASG) with EC2 Instances**

* **Service**: Auto Scaling automatically adjusts the number of EC2 instances based on traffic demands.
* **Reasoning**: ASG ensures high availability by dynamically adjusting the capacity of EC2 instances based on load. This helps handle spikes in traffic while optimizing costs by scaling down during low demand periods.

**1.4 Application Load Balancer (ALB)**

* **Service**: Amazon ALB distributes incoming application traffic across multiple EC2 instances or containers.
* **Reasoning**: ALB provides load balancing, health checks, and HTTP/HTTPS routing, ensuring that only healthy instances receive traffic. It supports high availability and fault tolerance for the application.

**1.5 Elastic Load Balancer (ELB)**

* **Service**: Internal ELB is used to load balance traffic between internal services in private subnets.
* **Reasoning**: ELB ensures efficient distribution of backend traffic between ECS services. It is deployed in private subnets to protect internal services from external access.

**1.6 ECS Fargate**

* **Service**: Amazon ECS Fargate is a serverless compute engine for containers.
* **Reasoning**: Fargate removes the need to manage EC2 instances for running containers. It automatically provisions the compute capacity for containers, making it cost-efficient and scalable. Fargate is used to run the API services.

**1.7 RDS (Relational Database Service)**

* **Service**: Amazon RDS with Multi-AZ deployment ensures database availability and fault tolerance.
* **Reasoning**: RDS manages complex tasks like backups, patching, and replication, reducing operational overhead. The Multi-AZ configuration ensures that a standby database is always available in case of failure, improving reliability.

**1.8 Security Groups**

* **Service**: Security Groups act as virtual firewalls to control inbound and outbound traffic at the instance level.
* **Reasoning**: Security Groups enforce strict traffic rules between resources (EC2, ALB, ECS, RDS), ensuring that only permitted communication takes place. This enhances security and minimizes attack surfaces.

**1.9 Amazon CloudWatch**

* **Service**: CloudWatch provides monitoring and observability for AWS resources.
* **Reasoning**: CloudWatch tracks metrics like CPU usage, memory, and network traffic for EC2, RDS, and Auto Scaling Groups. It triggers alerts and notifications based on predefined thresholds, improving operational visibility.

**2. Architecture Overview**

**High-Level Architecture**

* **VPC**: A VPC with two public and two private subnets distributed across multiple availability zones for high availability.
* **ALB**: The Application Load Balancer resides in the public subnets, routing incoming HTTP/HTTPS traffic to the EC2 instances in the Auto Scaling Group.
* **Auto Scaling Group**: EC2 instances are part of an Auto Scaling Group in the public subnets. The ASG dynamically adjusts capacity based on traffic patterns, ensuring that enough instances are running to handle incoming requests.
* **ECS Fargate**: The ECS Fargate cluster runs containerized APIs in the private subnets, ensuring backend services are secure and scalable without managing servers.
* **Internal ELB**: An Internal Elastic Load Balancer distributes traffic among ECS tasks, providing high availability and fault tolerance for backend services.
* **RDS**: The Multi-AZ RDS instance, located in private subnets, provides a highly available and fault-tolerant database for the API.
* **Security Groups**: Security Groups are used to control traffic between resources and ensure that only authorized access is allowed.
* **CloudWatch**: CloudWatch monitors the performance of the infrastructure, providing insights and alerting for resource utilization and health.

**Traffic Flow**

* Traffic enters the public subnets through the ALB, which forwards requests to the EC2 instances in the Auto Scaling Group.
* The EC2 instances in turn interact with the ECS Fargate cluster via the Internal ELB for backend API services.
* The RDS instance in private subnets handles data storage for the API, ensuring fault tolerance and high availability.

**3. Deployment Process with Terraform**

**3.1 Pre-Requisites**

* **AWS CLI**: Firstly AWS CLI have to be configured with the necessary credentials.
* **Terraform Installed**: Secondly have to install Terraform to manage the infrastructure as code.

**3.2 Terraform Files**

The infrastructure is organized into modules, and the following Terraform files are used:

* **main.tf**: This file orchestrates all the modules (VPC, Auto Scaling, ECS, RDS, Security Groups).
* **providers.tf**: Configures the AWS provider and region.
* **variables.tf**: Defines input variables for customization (region, instance type, subnet CIDRs).
* **outputs.tf**: Captures and outputs key information like VPC ID, ALB DNS, RDS instance ID.
  1. **Terraform Commands**

1. **Initialize the Terraform Working Directory**:

* This command downloads the necessary provider plugins and sets up the backend if using remote state.

*terraform init*

1. **Plan the Changes**:

* This command generates an execution plan, showing what resources will be created, modified, or destroyed.

*terraform plan -out=tfplan*

1. **Apply the Plan**:

* This command is used to apply the changes and provisions the infrastructure on AWS.

*terraform apply -auto-approve tfplan*

1. **Terraform Destroy**:

* This command is used to destroy all the resources managed by Terraform.

*terraform destroy*

**Conclusion:**

This architecture leverages AWS services like VPC, ALB, ECS, RDS, and Auto Scaling to build a highly available, fault-tolerant, and scalable infrastructure. The deployment process is automated using Terraform and Azure DevOps pipelines, ensuring that infrastructure changes are made consistently and efficiently. By following AWS best practices for security, reliability, performance, and cost optimization, this architecture is well-suited for production-grade.