**Deploying a Hub-and-Spoke 3-Tier Architecture on AWS**

A computer screen shot of a computer

AI-generated content may be incorrect.

**Overview**

In this project, we will architect and deploy a scalable and highly available **Hub-and-Spoke** model with a **3-Tier Application Deployment** on AWS. This architecture consists of **multiple VPCs (Dev, Staging, Pre-Prod, and Prod) interconnected via AWS Transit Gateway**. The solution is designed for security, scalability, and high availability across multiple **Availability Zones (AZs)** while providing efficient routing and controlled access to application components.

We will leverage key AWS services such as **CloudFront, Route 53, ALB, Auto Scaling Groups (ASG), EC2, RDS, and CloudWatch** to ensure optimal performance, fault tolerance, and seamless integration.

**Tool Stack Overview**

* **Route 53** – AWS’s scalable and highly available **Domain Name System (DNS)** service that routes users to applications by mapping domain names to IP addresses, AWS resources, or external endpoints.
* **CloudFront** – A **Content Delivery Network (CDN)** that caches and delivers content (static and dynamic) from edge locations worldwide, reducing latency and improving performance.
* **AWS Certificate Manager (ACM)** – A service that **provisions, manages, and renews SSL/TLS certificates** for securing AWS applications, commonly used with CloudFront, ALB, and API Gateway.
* **CloudWatch Alarm** – A monitoring tool that triggers **automated actions** (notifications, auto-scaling, stopping instances) based on predefined thresholds for AWS service metrics.
* **CloudWatch Logs** – A centralized logging service that **collects, monitors, and stores logs** from AWS services and applications, allowing real-time monitoring, debugging, and analysis.
* **RDS MySQL** – A **managed relational database service** for MySQL that automates administrative tasks like backups, scaling, and patching, offering high availability.
* **EC2 (Elastic Compute Cloud)** – AWS’s virtual machines that provide **scalable compute capacity** in the cloud, supporting various OS and application workloads.
* **Application Load Balancer (ALB)** – A **Layer 7 load balancer** that distributes HTTP/HTTPS traffic across multiple EC2 instances, containers, or Lambda functions, supporting advanced routing features.
* **NGINX** – A **high-performance web server and reverse proxy** used for load balancing, caching, and optimizing web traffic for scalability and security.
* **PM2** – A **process manager for Node.js applications** that ensures applications remain online, restarts them on failure, and provides monitoring tools.
* **Node.js** – A **JavaScript runtime** that allows developers to build scalable, high-performance web applications by running JavaScript on the server.
* **React** – A **JavaScript library** for building interactive user interfaces, primarily used in **Single Page Applications (SPAs)** with a component-based architecture.
* **Transit Gateway** – A **network hub that connects VPCs, on-premises networks, and AWS services** at scale, simplifying routing and security across multiple AWS environments.
* **Site-to-Site VPN** – A **secure connection** between an on-premises network and AWS VPC using **IPsec tunnels**, enabling hybrid cloud connectivity and secure data transfer.
* **S3 Bucket** – AWS’s **object storage service** that offers scalability, security, and durability for storing static files, backups, logs, and hosting static websites.

**Defining the AWS Cloud Environment**

We will begin by defining the AWS cloud environment as the foundation for our solution. Within this environment, we’ll make use of the **Hub-and-Spoke** model that consists of multiple VPCs, each dedicated to a specific environment (**Development, Staging, Pre-Production, and Production**). These VPCs are connected via **AWS Transit Gateway** to enable secure, controlled communication across environments.

Low level environment contains each two Private subnets (one per availability zone), however the top level environment will contain **three subnets** per Availability Zone:

1. **Web Subnet** (Public) – Hosts a **React application** running on a **NGINX server** handling external requests.
2. **App Subnet** (Private) – Runs the application layer will host a **Nodejs application**, managed with PM2 for efficient process management.
3. **Database Subnet** (Private) – Houses MySQL RDS instances for secure data storage.

**Data Tier (Database Layer)**

* I’ll deploy a**n RDS MySQL** in the **Prod-VPC** within the **Database Subnet** for secure data management.
* The **primary RDS instance** will run in one Availability Zone, with a **standby instance** in another for failover support.
* Security Groups will restrict access, allowing only the **application tier** to communicate with the database over **port 3306**.
* I’ll implement a **multi-AZ setup** to ensure high availability and business continuation in case of failure or disruption in the primary.

**Application Tier**

* To dynamically scale our application tier to meet fluctuating demands, we’ll implement an application tier **autoscaling group**, this will strategically manage EC2 instances across the multiple availability zones **optimizing the resources utilization**
* By deploying our application tier instances within this auto scaling group, we’ll enable the platform to automatically **adjust capacity** based on real time requirements
* Each EC2 instance will host our **NodeJS application** managed by **PM2** for efficient process management.
* This approach empowers our application tier to **handle varying workload** effectively, **preventing performance bottlenecks** and ensuring a seamless user experience.

**Presentation Tier**

* To manage and scale our **presentation tier** effectively we’ll implement an **autoscaling group**. This dynamic infrastructure component will distribute presentation tier EC2 instances across multiple availability zones ensuring **high availability and performance.**
* To securely access and manage these instances, we’ll deploy **a bastion host** outside the autoscaling group.
* This bastion host will serve as a **control entry point** to our **private network** allowing us to manage and troubleshoot those instances as needed.
* Each instance within this presentation tier autoscaling will function as a **presentation tier endpoint**, utilizing **NGINX** as a webserver to deliver our **ReactJS** application to end-users.
* To efficiently distribute traffic within our application tier, we’ll implement an **internal application load balancer.**
* This load balance will direct traffic across multiple EC2 instances within the private subnets.
* To manage external traffic, we’ll deploy an **internet facing application load Balancer**, distributing incoming requests across the presentation tier instances
* This two-tier Load Balancing approach enhances scalability and performance.

**Inter-VPC Communication & Routing**

* I’ll leverage the distributive ability of the **Transit Gateway (TGW)** to enable **secure connectivity** between the different environments, **Dev, Staging, Pre-Prod, and Prod VPCs**.
* I’ll not make use of a **VPC Peering,** to reduce complexity and improve scalability.
* Each VPCwill have **a route table** directing traffic either **locally** or to the **Transit Gateway (TGW)**.
* I will implement a **Site-to-Site (S2S) VPN** to establish secure connectivity between the **on-premises network** and other environments

**Security & Access Controls**

* The Web **Security Group** will be configured to allow only **HTTP (80)** and **HTTPS (443) traffic**.
* Application Security Group allows communication from Web Servers to Application Servers.
* To ensure secure communication, the **Application Security Group** will permit traffic only from **Web Servers to Application Servers**.
* Database Security Group restricts access to only the Application Tier.
* Access to the **Database Tier** will be restricted by implementing a Database Security Group that **only** allows connections **from the Application Tier**.
* A Bastion Host Security Group will be set up to provide controlled SSH (22) access for secure maintenance operations.

**End-User connection**

* To effectively manage domain name resolution and ensure secure communication we’ll leverage amazon **Route53 for DNS management**.
* By creating **DNS records** within Route53, we’ll **map our domain** to the appropriate resources.
* To establish a secure **HTTPS connection**, we’ll procure an **SSL Certificate** from **AWS Certificate Manager**
* This certificate will be integrated with **CloudFront**, our **content delivery network**, to **encrypt data** in **transit** and optimize content delivery to end-users.
* With all components in place, User can securely access our application via HTTPS, our database architecture ensures high availability through synchronous replication of data between primary and standby instances, in the event of a failure amazon RDS automatically transition to the standby instance minimizing downtime and data loss.

**Monitoring & Observability**

* To enhance observability and optimize resource utilization, we’ll integrate CloudWatch into our architecture.
* Application logs will be centrally managed within CloudWatch logs, providing easy access for troubleshooting and performance analysis.
* CloudWatch alarms will be configured to trigger scaling action within the autoscaling group, ensuring optimal resource allocation based on demand.
* By scaling out during peak usage and scaling in during low traffic periods we’ll maintain cost efficiency while delivering consistent performance.

**Deployment Workflow**

**Step 1: Configure Route 53 & CloudFront**

* Set up **Route 53 Hosted Zone** and configure the DNS records
* Create **aliases** or **CNAME records** to route user traffic to **CloudFront** and the **Application Load Balancer (ALB)**.
* Attach **AWS Certificate Manager (ACM) SSL/TLS certificates** for secure HTTPS communication.
* Validate the domain ownership through DNS or email validation
* Configure **CloudFront** to serve content and cache static files efficiently.
* Attach the certificate to services like CloudFront or ALB for end-to-end encryption

**Step 2: Create & Configure VPCs**

* Define all the environments: **Dev, Staging, Pre-Prod, and Prod VPCs**, each with **CIDR ranges**.
* Deploy **subnets (Web, App, DB) across multiple AZs** for fault tolerance.
* Configure route tables, internet gateways, and NAT gateways to ensure proper connectivity between tiers.

**Step 3: Transit Gateway Setup**

* create a Transit Gateway (Prod-TGW), Enable Auto Accept Shared Attachments if connecting multiple accounts
* create an attachment for each VPC (Dev, Staging, Pre-Prod, Prod).
* Update Route Tables. In each VPC Route Table, add routes: Other VPC CIDRs → Target: Transit Gateway (TGW-ID)
* Enable Site-to-Site VPN, Attach VGW or create a new TGW VPN attachment to allow on-prem connectivity. Update TGW Route Table to route on-prem CIDR to VPN Attachment.
* Modify Security Groups & NACLs, Allow inter-VPC traffic for required services (e.g., HTTP, HTTPS, RDS, SSH).

**Step 3: Configure Site-to-Site VPN**

* Create **Virtual Private Gateway (VGW)** and attach it to Prod-VPC
* Create **Customer Gateway (CGW)** and add **on-prem public IP** as a CGW in AWS.
* Create **VPN Connection**, configure VPN with VGW & CGW, choose Static or BGP routing
* Download VPN settings from AWS and apply them to on-prem firewall.
* In Prod-VPC Route Table, add a route for on-prem CIDR (192.168.1.0/24) → VGW.
* Allow traffic from on-prem IPs for SSH (22), RDS (3306), and required services.
* Test Connectivity by checking VPN Tunnel Status (UP) in AWS, Ping AWS resources from on-prem and vice versa
* Set up CloudWatch Alarms for VPN Tunnel State to track downtime.

**Step 3: Configure Security & Access Management**

* Define **IAM roles and policies** to enforce least-privilege access.
* Define security groups to **control inbound** and **outbound traffic.**
* Implement **Security Groups & Network ACLs** to restrict access appropriately.
* For example, restrict database access to application instances only and allow traffic to the ALB from the internet

**Step 4: Launching the Bastion Host**

* Deploy an EC2 instance in the public subnet to act as the bastion host for securely accessing private resources (e.g., RDS and application servers).
* Configure SSH access to the bastion host and enable session management with tools like AWS Systems Manager for enhanced security.

**Step 5: Setup the Database Tier (RDS MySQL)**

* Launch an **RDS MySQL instance** in the **Prod-VPC DB subnet**.
* Configure database **security groups** to allow access only **from the application tier**
* Set up **automated backups**, **multi-AZ replication**, and **encryption** for high availability and data protection.

**Step 6: Setup the Application Tier**

* Launch **EC2 instances** in the **Prod-VPC App subnets**.
* Install **Node.js & PM2** for efficient application process management.
* Configure **Auto Scaling Groups (ASG)** to handle traffic fluctuations.

**Step 7: Setup the Web Tier (NGINX & ALB)**

* Deploy **NGINX Web Servers** in the **Prod-VPC Web subnets**.
* Set up an **External ALB** to route incoming requests to **Web Servers**.
* Configure **Internal ALB** to distribute traffic from **Web Servers to Application Servers**.
* Configure target groups, health checks, and routing rules to ensure efficient request handling.

**Step 8: Configure Monitoring & Logging**

* Configure **CloudWatch Logs** to collect application, server, and database logs for real-time monitoring and troubleshooting.
* Set up CloudWatch Alarms to notify the team of critical events or performance issues
* Enable **CloudTrail** for API activity logging.

**Step 9: Creating a CloudFront Distribution**

* Configure AWS CloudFront as the content delivery network (CDN) for the application, caching static assets for low-latency delivery.
* Integrate the distribution with the ALB or directly with the S3 bucket (if hosting static files).

**Step 10: Validate and Test the Deployment**

* Add CNAME or alias records in Route53 to map the domain to the CloudFront distribution.
* Perform **DNS resolution tests** using Route 53.
* Validate **Load Balancer routing** and **Auto Scaling behavior**.
* Conduct **failover tests** for **RDS Multi-AZ & EC2 Auto Scaling**.

**Conclusion**

This Hub-and-Spoke 3-Tier Architecture ensures a secure, scalable, and highly available environment for production-grade applications. By leveraging AWS Transit Gateway, Route 53, CloudFront, ALB, ASG, EC2, and RDS, we provide a robust cloud infrastructure capable of handling dynamic workloads efficiently.