Services Required:

## **1. Frontend (Nuxt.js) Deployment**

* **Option 1: AWS Amplify**
  + Easiest for static sites or SSR, integrates with GitHub, and automates builds & deploys.
  + Great for beginners.
* **Option 2: Amazon S3 + CloudFront**
  + For static Nuxt builds (nuxt generate), host on S3 as a static website, use CloudFront for CDN.
  + No server-side rendering.
* **Option 3: Amazon EC2 or AWS Elastic Beanstalk**
  + For full SSR/Node.js hosting, run Nuxt in “server” mode. Use EC2 (virtual server) or Elastic Beanstalk (PaaS for web apps).

**Recommendation:** Start with Amplify for static or simple SSR; use EC2/Beanstalk for advanced server hosting.

## **2. Backend (Rust) Deployment**

* **Amazon EC2**
  + Most common: launch a Linux EC2 instance, install your Rust app, and run it as a service.
* **(Advanced) AWS Lambda**
  + For serverless, but Rust support is more complex for beginners.

**Recommendation:** EC2 is the simplest starting point.

## **3. Database (Postgres)**

* **Amazon RDS (Relational Database Service)**
  + Managed Postgres database (no need to install/patch manually).
  + Secure, scalable, automated backups, easy to connect from EC2 or backend.

## **4. Other Essential Services**

* **VPC (Virtual Private Cloud):**
  + Networking for your resources (default VPC is fine for small projects).
* **IAM (Identity and Access Management):**
  + Control permissions for AWS resources.
* **Route 53:**
  + DNS service for custom domains.
* **Certificate Manager:**
  + Free SSL certificates for HTTPS.

**Services selected**

* FrontEnd: AWS Elastic bean stack
* Backend: EC2
* Database: Amazon RDS
* VPC
* IAM
* Route 53
* Certificate manager

## **High-Level Architecture Components**

### **1. VPC (Virtual Private Cloud)**

* Create a **custom VPC** with:
  + 2 Public Subnets (for Load Balancer / Elastic Beanstalk)
  + 2 Private Subnets (for EC2 + RDS)
  + Internet Gateway (for public access)
  + NAT Gateway (for EC2 instances to access the internet)
  + Route Tables configured for public/private traffic routing

### **2. Frontend (Nuxt.js) - Elastic Beanstalk**

* Use **AWS Elastic Beanstalk (Nginx/Node.js platform)** for easy deployment and scaling.
* Deploy your Nuxt static files using SSR or static generation.
* Configure **Auto Scaling** and **Load Balancing** in Beanstalk.
* Attach **SSL certificate** from ACM (AWS Certificate Manager).
* Use **Route 53** to map your domain to the Beanstalk environment.

### **3. Backend (Rust Actix-Web) - EC2**

* Launch **EC2 instances** in **private subnets** behind a **Network Load Balancer (NLB)** or **Application Load Balancer (ALB)**.
* Use **Amazon EC2 Auto Scaling Groups** for horizontal scaling.
* Backend listens on internal IPs (only accessible via Load Balancer).
* Secure with **Security Groups** and **IAM roles**.

### **4. Database - Amazon RDS (PostgreSQL)**

* Deploy **PostgreSQL** in private subnets.
* Enable:
  + Multi-AZ for high availability
  + Automatic backups
  + Performance insights
  + Encryption at rest and in transit
* Restrict access via security group (only backend EC2s can connect)

### **5. IAM (Identity & Access Management)**

* Create **IAM roles** for:
  + EC2 (to access S3, CloudWatch logs)
  + Elastic Beanstalk (to manage EC2, S3, etc.)
  + RDS (to enable monitoring/metrics)
* Use **least privilege principle** for IAM policies.

### **6. Route 53 (DNS)**

* Use **hosted zones** for your domain.
* Point subdomains to:
  + Frontend → Beanstalk CNAME
  + Backend API → Load Balancer DNS name

### **7. Certificate Manager (SSL)**

* Use **AWS ACM** to provision free SSL certificates.
* Attach to ALB and Beanstalk for HTTPS.

### **8. AWS CloudWatch**

* For **centralized logging, metrics, alarms**.
* Configure logging for:
  + Beanstalk frontend
  + EC2 backend
  + RDS
* Set alarms for CPU, memory, disk, etc.

### **9. AWS S3 (optional)**

* For hosting static assets/images
* Use **CloudFront CDN** to distribute assets globally.

### **10. AWS Secrets Manager**

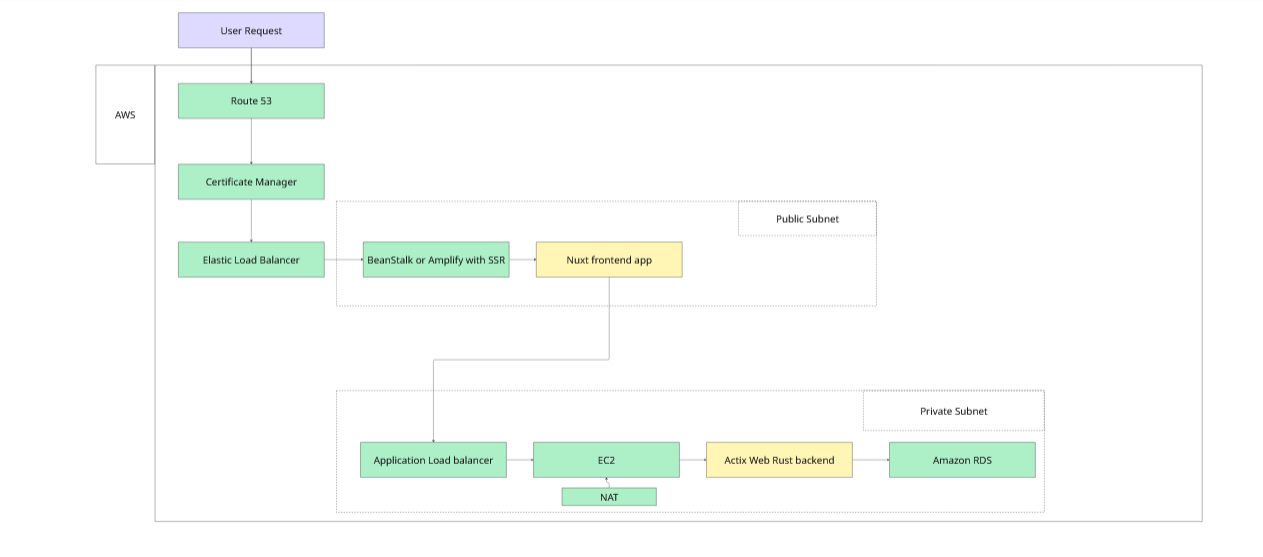
* Securely store **DB credentials**, **API keys**, etc.
* Access secrets programmatically in EC2/backend.

### **11. AWS CodePipeline + CodeBuild *(Optional CI/CD)***

* Automate deployments to Beanstalk & EC2.
* Connect to GitHub or CodeCommit.

## **Security Measures**

* **Use HTTPS** everywhere (ACM + ALB/Beanstalk).
* **Private subnets** for EC2 & RDS.
* **Security Groups** to tightly control access.
* **IAM Roles** with strict permissions.
* Use **CloudTrail** to audit all actions on AWS.
* Enable **RDS encryption**, **backup**, and **failov**

**Architecture:**  
  


## **SECURITY FLOW**

* **Public Subnets**:
  + Elastic Load Balancer (frontend entry point)
  + NAT Gateway
  + Route 53, ACM
* **Private Subnets**:
  + EC2 backend (Actix)
  + Nuxt SSR server (if not using static generation)
  + Amazon RDS (PostgreSQL)

## **RESOURCES MAPPED TO SERVICES**

|  |  |  |
| --- | --- | --- |
| **Component** | **AWS Service** | **Notes** |
| DNS | Route 53 | Maps domain to ELB |
| SSL/TLS | ACM | Auto-renews certificates |
| Frontend (NuxtJS) | Elastic Beanstalk | Static/SSR deployment |
| Backend (Rust/Actix) | EC2 + Auto Scaling Group | Scales backend containers |
| Database | Amazon RDS (PostgreSQL) | Encrypted, Multi-AZ |
| Load Balancing | Application Load Balancer | For EC2/Beanstalk |
| Networking | VPC + Public/Private Subnets | Isolate resources |
| NAT Gateway | NAT Gateway | Lets private instances access internet |
| IAM Roles & Policies | IAM | Restrict access via least privilege |
| Monitoring & Logs | CloudWatch | For metrics and logs |
| Secrets Storage (opt) | Secrets Manager | Store DB/API credentials securely |
| CI/CD (optional) | CodePipeline + CodeBuild | Automated deployments |

## **RECOMMENDED DEPLOYMENT FLOW**

1. **Setup VPC**: 2 public + 2 private subnets across AZs.
2. **Deploy Frontend**: Elastic Beanstalk with Node.js for Nuxt SSR or static build.
3. **Deploy Backend**: EC2 instances using AMI or Docker via Auto Scaling group.
4. **Provision RDS**: PostgreSQL with backups, failover, security groups.
5. **Configure Load Balancer**: Route frontend/backend traffic to right targets.
6. **Set DNS via Route 53**: Map domain → ELB endpoints.
7. **Attach SSL via ACM**: Use HTTPS everywhere.
8. **Monitor**: Set CloudWatch alarms for CPU, RAM, and logs.
9. **Secure**: IAM policies, VPC flow logs, no open DB/public EC2 access.

**Request journey:**  
  
**1. User opens your website (e.g.,** [**https://www.powerbrilliant.com**](https://www.powerbrilliant.com)**)**

* **Service involved**: **Route 53**
* **What happens**:
  + DNS query resolves [www.powerbrilliant.com](https://www.powerbrilliant.com) to the Elastic Load Balancer's public IP.

#### **2. HTTPS request hits Load Balancer**

* **Service involved**: **Elastic Load Balancer (ALB) + ACM**
* **What happens**:
  + TLS/SSL is terminated using the certificate from **Certificate Manager (ACM)**.
  + Load Balancer routes the request to the **Elastic Beanstalk** environment hosting your Nuxt frontend.

#### **3. Frontend processes request**

* **Service involved**: **Elastic Beanstalk (Node.js / Nuxt SSR)**
* **What happens**:
  + Nuxt handles the rendering of the login page or prefetches data.
  + On form submission, Nuxt sends a **POST /api/login** request to your **Actix backend**.

#### **4. Request goes to Backend**

* **Service involved**: **ALB → EC2 instance (Actix Web API)**
* **What happens**:
  + The load balancer forwards /api/\* paths to backend target group.
  + Actix receives the login request and processes credentials.
  + If password is hashed or JWT is involved, computations happen here.

#### **5. Backend accesses the Database**

* **Service involved**: **Amazon RDS (PostgreSQL)**
* **What happens**:
  + Actix sends a query to RDS in a **private subnet** (e.g., SELECT \* FROM users WHERE email=?).
  + RDS processes the request and returns result securely to the backend.

#### **6. Backend returns response**

* **Service involved**: **EC2 instance (Actix Web API)**
* **What happens**:
  + Backend returns login success/failure and issues JWT or session token.

#### **7. Frontend updates UI**

* **Service involved**: **Nuxt.js**
* **What happens**:
  + Nuxt stores JWT or session info and navigates user to their dashboard.
  + Data may now load using API requests to the backend.

#### **8. Static assets served (optional)**

* **Optional Services**: **S3 + CloudFront**
* **What happens**:
  + Any images, JS/CSS, or assets are fetched from S3 with CDN via CloudFront if configured.