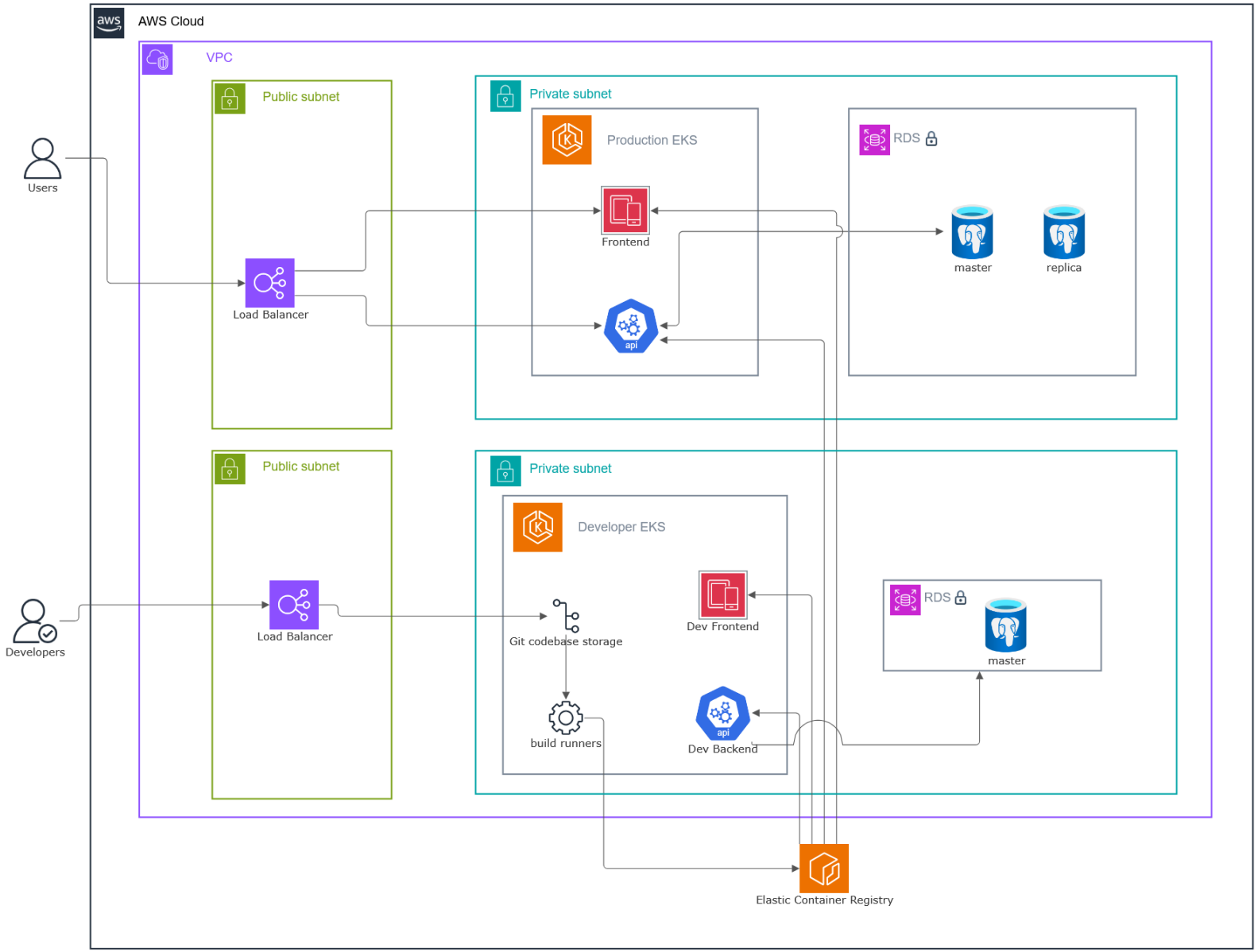
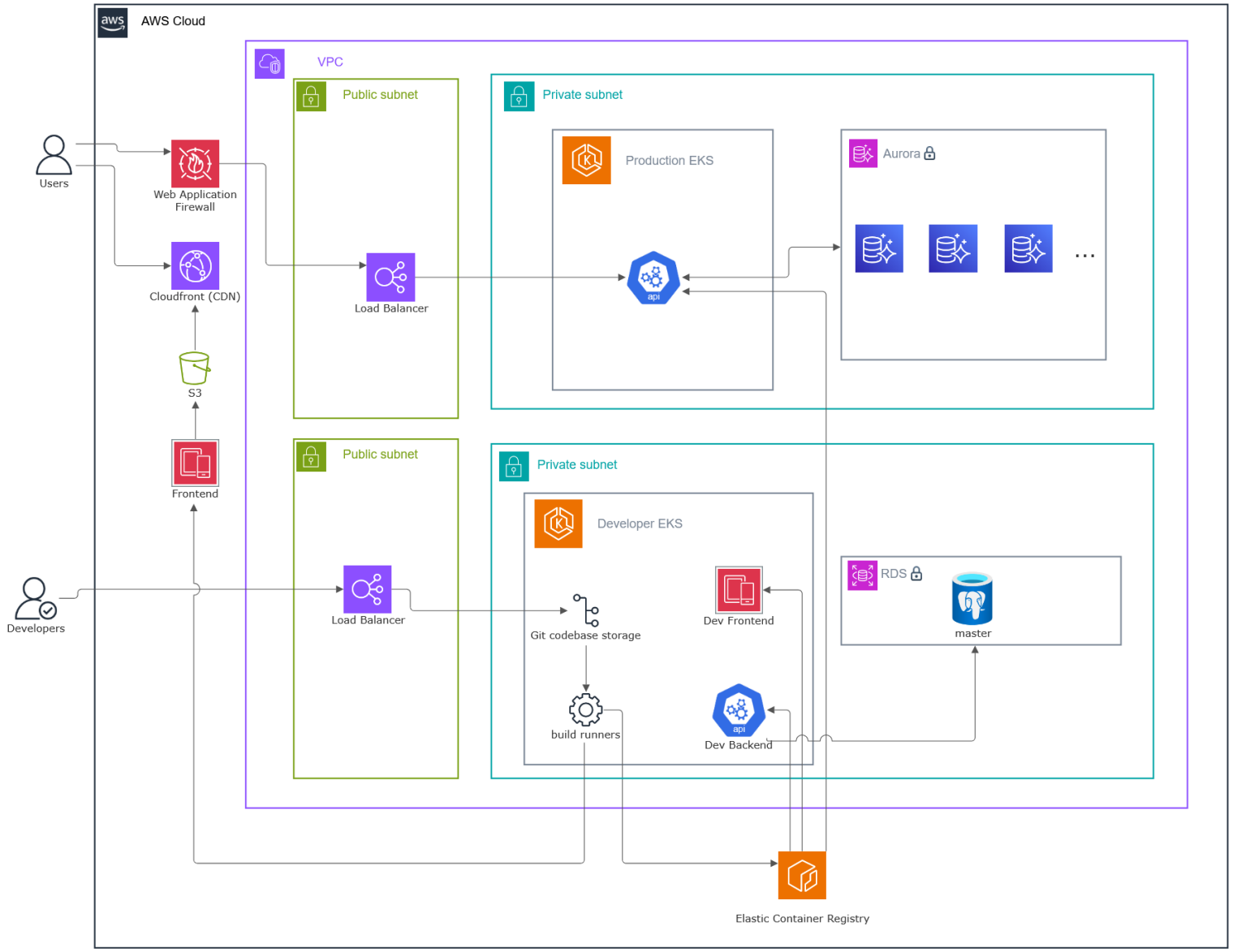
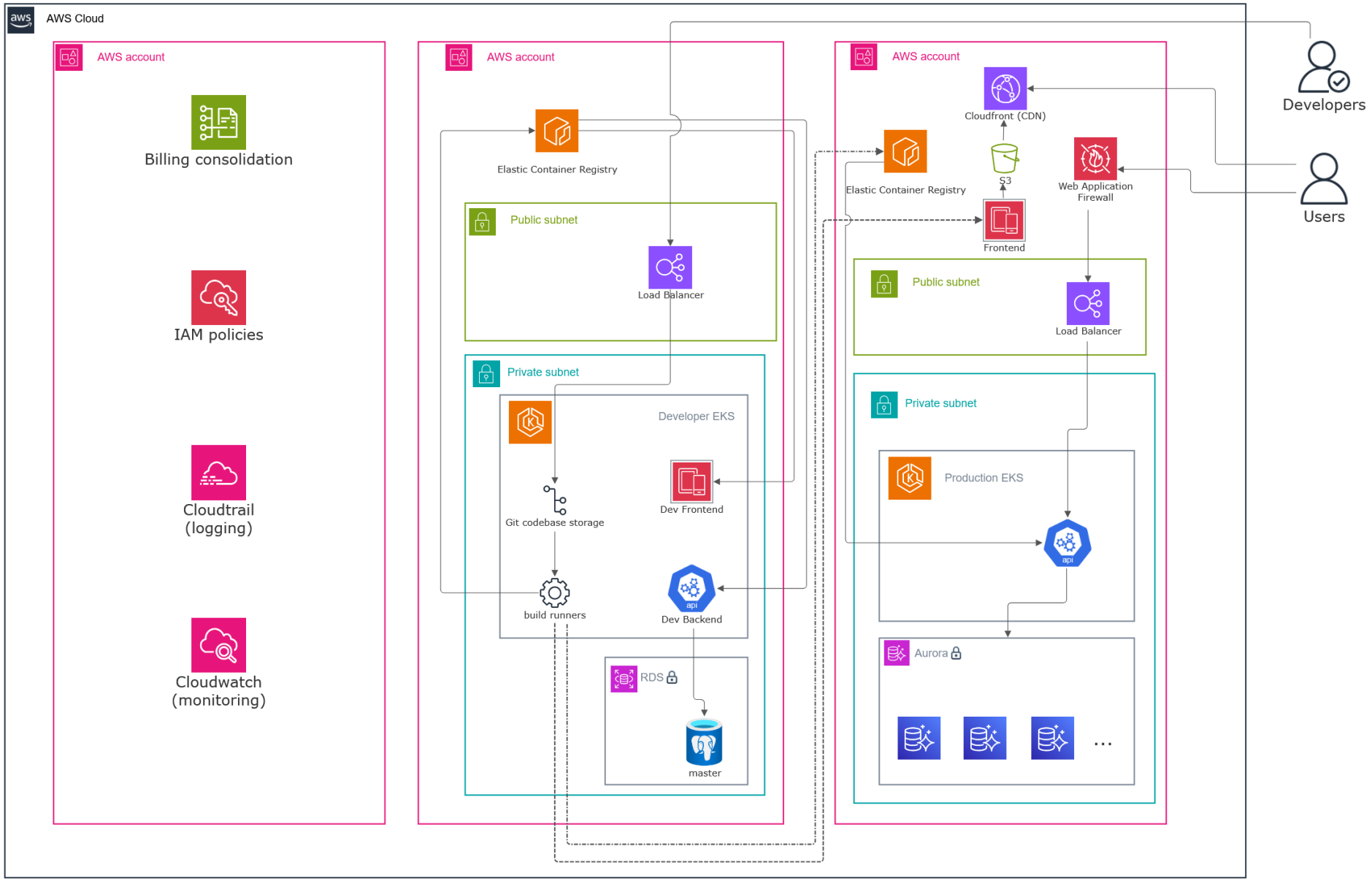
Option 1. Simple Starter



Option 2. Grown project



Option 3. All best-practices



Option 1. Simple Starter. General

Dev and prod environment are separated in different subnets

All workload are in private subnet, accessible only from Load Balancer and its security group

Option 1. Simple Starter. Prod

Frontend are deployed in EKS cluster. Should be efficient while we have under 50k users daily.

Postgres are deployed in RDS. Have all necessary features (replicas, backups, sufficient compute capacity), doesn`t cost as much as Aurora.

Data in RDS are encrypted both at-rest and in-transit.

Prod uses shared Elastic Container Registry, internally segregated with naming or tagging conventions.

Option 1. Simple Starter. Dev

Frontend are deployed in EKS cluster. No need to allocate additional resources for it

Postgres are deployed in separate RDS, for isolation

Data in RDS are encrypted both at-rest and in-transit.

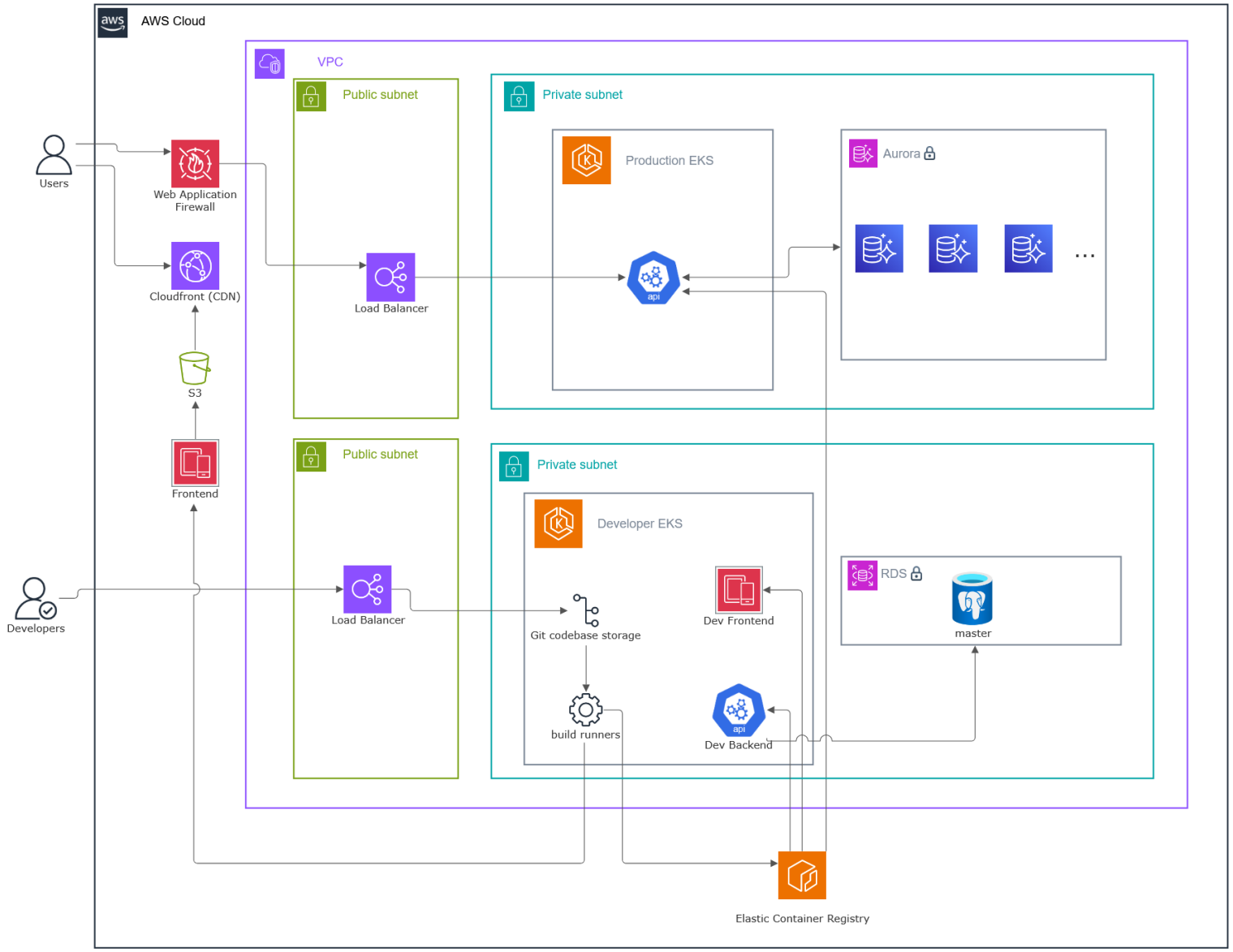
In Dev EKS Git codebase repo (Gitlab, Bitbucket, etc) are deployed, if needed, as well as Build runners (Gitlab CI, Jenkins, Github Actions runner).

Dev uses shared Elastic Container Registry, internally segregated with naming or tagging conventions

Option 1. Simple Starter.

Good as starter and POC. Cheap, easy to deploy and easy to use

Option 2. Grown project



Option 2. Grown project. General

Similar to option 1, further developed

Dev and prod environment are separated in different subnets

All workload are in private subnet, accessible only from Load Balancer and its security group

Option 2. Grown project. Prod

Frontend has been moved to Cloudfront(CDN) for offload and faster user access time

Backend has been moved behind Web Application Firewall for enhanced security

Postgres has been moved to Aurora. Autoscaling both replicas and disk size, Multi-AZ storage, greater number of possible replicas. More expensive than RDS, but preferable for prod.

Data in Aurora are encrypted both at-rest and in-transit.

Prod uses shared Elastic Container Registry, internally segregated with naming or tagging conventions.

Option 2. Grown project. Dev

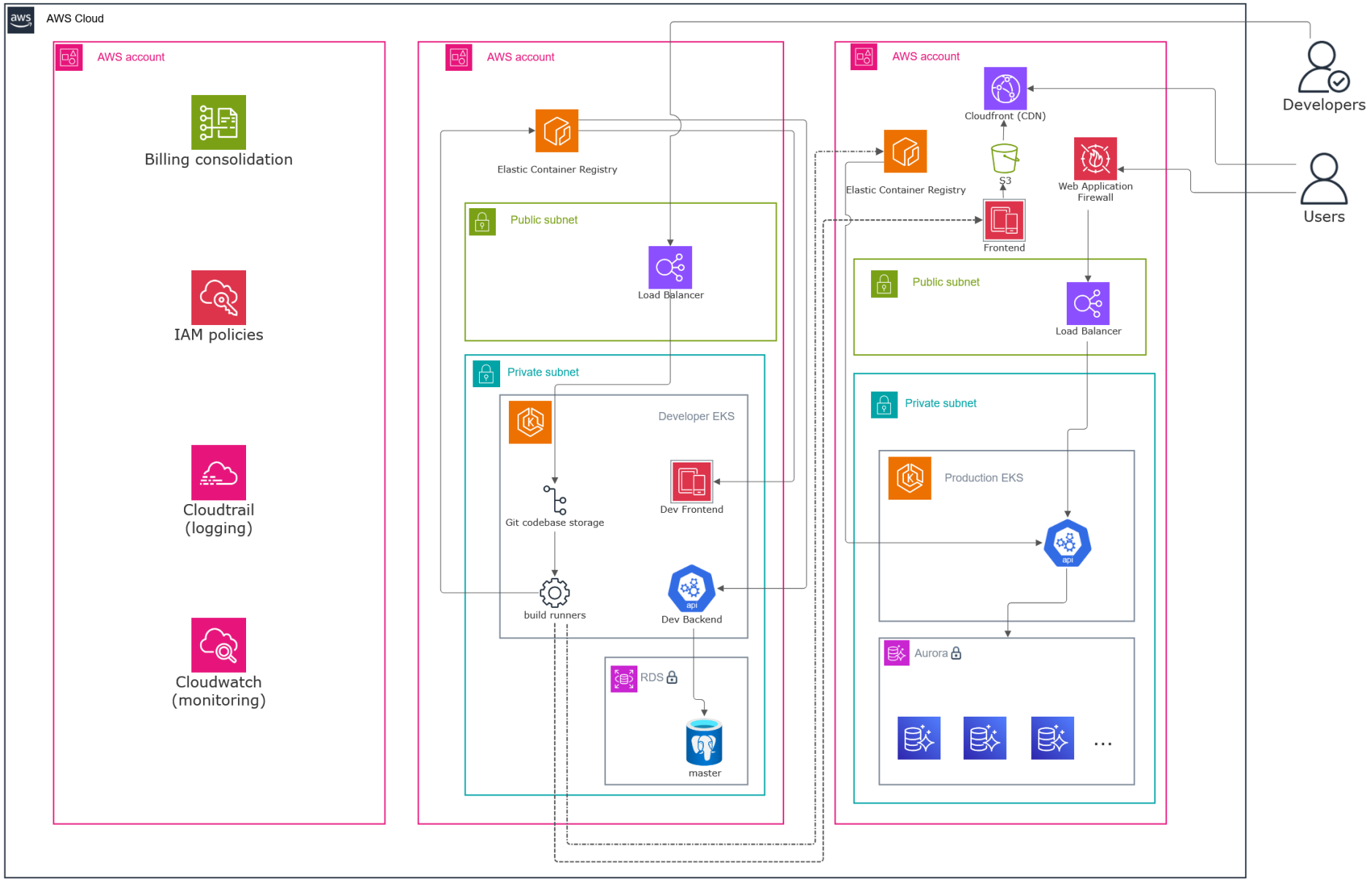
No changes from starter variant.

Option 2. Grown project.

This architecture meets security compliance, remaining relatively simple to work with for developers and operations.

Access to prod resources are regulated via IAM roles and groups, both for clients and developers.

Option 3. All best-practices



Option 3. All best-practices

AWS Organizations is set up, with 3 AWS accounts

Max compliant and more complicated for operations and developers.

May cost more directly (additional ECR, etc) and indirectly (more complicated -> more time required for operations` tasks)

Option 3. All best-practices. Managing account

Holds higher-level IAM policies.

Center of SSO

Billing consolidation in one place

Cloudwatch (monitoring) consolidation in one place

Cloudtrail (logging) consolidation in one place

Generally, your control room

Option 3. All best-practices. Dev account

Holds Dev/Staging environment, fully isolated from prod

Separate Elastic Container Registry

Similar to dev environment in previous options

Option 3. All best-practices. Prod account

Holds prod environment, fully isolated from dev

Separate Elastic Container Registry

Similar to prod environment in option 2