Worktest

General Architecture

The system consists of a public eks cluster running on a vpc with 4 subnets (2 private and 2 public) inside 2 availability zones. The workload comprises a managed node group running 2 t3.medium on-demand ec2 instances.
I choose eks to keep things as cloud-agnostic as possible for any future multi-cloud migration.

Inside the created vpc, we both have public and private subnets. Even though the cluster only uses public ips in a production environment, private ips are not needed and would be a waste if the cluster is public only, but I included them with Terraform to make some possible changes easier while developing.

I chose ec2 instead of fargate for computing resources, mainly because of the ebs requirement for the monitoring setup. The system could also work with an additional fargate node group just labeled for apps, but it would be a waste of resources with this specific demo (yes, I know technically fargate supports ebs, but it has only been one year)

With eks, we have an aws-ebs-csi-driver add-on other than the standard networking add-ons. This was required for ebs integration.

Management

Terraform is used for AWS resources like eks, vpc, and node groups, and the state is stored in an S3 bucket.

Official helm charts for ingress controllers, grafana, and prometheus are used with custom values, and deployment is done via the Terraform helm provider.

Prometheus and grafana helm charts are used in monolith modes without scaling features on production. This is discouraged

The mock app is not part of any Terraform templates or helm chart and needs to be deployed by hand; this is a deliberate choice because using Terraform for app deployments could count as an antipattern recommendation on a production environment would be a system like fluxcd, argo or any desired gitops service with kubernetes focus and good integration with role-based access controls.

IAM

We are directly adding policies to newly created ec2 instances for access control. They are used for ebs deployment, cloudwatch access, and elb deployment for the ingress controller.

AmazonEBSCSIDriverPolicy // a managed policy created for aws precisely to work with ebs CSI driver, allows anything running on the ec2 to create and destroy ebs volumes.
AmazonGrafanaCloudWatchAccess // a managed policy generally used for managed grafana service from aws to allow access for cloudwatch. Even though we are not using managed grafana, it still provides the necessary permissions to get logs.
ALBIngressPolicy // allows ec2 to create and destroy load balancers with any additional required resources.

This solution is not secure and should not be used in a production environment

- Some of the specified resources are too broad and allow ec2 to delete some unmanaged resources
- It doesn't allow pod-based access control, breaking the principle of least privilege

To solve this issue in a production eks environment, we would need IRSA or Pod identity agent.

Dashboards

We have six dashboards from https://github.com/dotdc/grafana-dashboards-kubernetes showing data about the general status of the kubernetes cluster and connected to the prometheus datasource

two from monitoringartist used to visualize usage, latency, and response status data from elb alb and classic instances and connected to cloudwatch datasource

one from kubernetes shows nginx-ingress metrics labeled by hostname connected to prometheus

These dashboards are supplied to helm chart with a values.yaml file, this way of provisioning is useful as day 1 deployment, but in the future, using configmaps for this purpose is a better solution

Ingress

We have two controllers for ingress to show different possible integrations with load balancers. They are :

- Nginx ingress controller // creates an elb classic instance in the background and routes requests to a nginx pod
- alb ingress controller

Pricing

Pricing without any data transfer and storage usage :

- EKS $0.1/h
- EC2 (2 t3.medium) $0.0944/h
- ELB ALB $0.02646/h
- ELB Classic 0.0294/h
- VPC NAT Gateway (2) $0.1/h
- VPC EIP (2) $0.01/h

Production & Scaling

Cloud-agnostic kubernetes

Even though EKS allows multi-cloud setup with hybrid nodes, it's never the best option since it

- Restricts usage of some eks add-ons
- Adds additional costs
- Forces managed aws networking tools for multi-cloud access, adding more costs
- Increases outbound data transfer

Because of these reasons, Hybrid eks is only recommended when it's being used with on-prem servers.

Depending on the expected cluster complexity, we can either use a 3rd party kubernetes control plane, manually deploy the control plane on servers, or use a multi-cluster approach.

3rd party control plane

With this option, we have some possible solutions with smaller cloud providers Scaleway, Digital Ocean, and Linode could count as good candidates they all are mostly lightweight planes without many vendor special services and we can register any of them to aws as an external cluster but a possible downside with this option would be the availability since they are not reliable as bigger cloud providers like AWS or azure.

Manual control plane deployment

Functionally, this is most similar to using a third-party plane with lesser costs. We could achieve high availability by working with bigger cloud providers, but it would cost more time and require more research and planning beforehand.

Multi-Cluster

Since every big cloud provider has additional control plane costs, this is the most expensive option but could be the easiest way to create a primarily resilient system. Usually, Rancher or a similar tool is used to manage a system like this.

Metrics

Prometheus is used as a metrics backend in this example, and it's a standard choice on prod environments as well as on a kubernetes cluster. Service discovery is done by labeling compatible services, and various exporters could be used as sidecar containers to enable metrics. It also has opentelemetry support, which we can enable if needed.
To scale prometheus, we can enable federation mode and use multiple instances per region or cloud provider.

If we have a cloud-managed or resilient database system, Grafana can be set in a high availability mode and use multiple instances with the same db

Logging

Log Backend

As the logging backend, we can use Loki. It's the standard tool used with grafana stack and easy to set up on kubernetes as a scaleable cluster with the helm for log storage. Using an external storage system is the better solution from an availability standpoint, and loki supports s3 compatible services, so we can use minio or cloudflare r2 as cheaper options.

Trace Backend

Of course, tracing depends on some other requirements from the actual running app, but if we have a traceid we can use, we can use tempo as the tracing backend, similarly to loki. Tempo also has monolithic and distributed deployment options with helm charts and options to store data on any s3 compatible platforms.

Collectors

- Fluentbit

setting the required config and adding a fluentbit sidecar to any desired pod would be enough to send logs to loki or any other popular log aggregation service or some cloud-managed services like data stream or s3

- Opentelemetry collector

Since prometheus and loki are compatible with opentelemetry, we can use their various exporters to collect metrics and logs and send them to our monitoring system. We can also use the opentelemetry operator to inject sidecars automatically instead of manually adding them to pods if needed.

Alerting

- Grafana

This is the easiest option and allows us to use multiple data sources, could achieve high availability depending on the grafana setup, and comes with integrations for many popular contact points; however, it could be harder to create provisioned alerts if there isn't a stable foundation for deployment management.

TEE

If we look into current big cloud providers, they currently have ways to add node groups with required hardware support to at least their own clusters

- https://docs.aws.amazon.com/enclaves/latest/user/kubernetes.html
- https://learn.microsoft.com/en-us/azure/aks/use-cvm
- https://cloud.google.com/blog/products/identity-security/announcing-general-availability-of-confidential-gke-nodes

When we look into more cloud-agnostic solutions, the main project supported with cncf is confidential containers (CoCo), which supports AMD-SEV, Intel SGX, and TDX. This covers use cases with gcp Azure and other smaller cloud providers; however, aws nitro enclaves could create problems, and the project is still in an early stage, so we should be careful and check the most recent features before using them on production.