

ACME Review

XX/XX/XXXX

Hans Knecht
Knechtions Consulting

Summary

This represents a comprehensive assessment of the AWS and EKS infrastructure, Github Configuration, and CI/CD practices. It also provides a variety of guidance and action items broken down into various sections. Each action item is also marked with easy, medium, or hard which denotes a combination of Level of Effort and Time required to complete. Where possible/applicable links are provided either to tools or to guides to enable self-directed completion of the work.

Key Areas of Focus

1. General AWS Configuration
2. EKS Configuration
3. Continuous Integration & Developer Experience
4. Continuous Deployment & Production Engineering
5. Application Architecture

AWS

I. IAM

- A. (Easy) - Use groups to assign permissions, don't assign directly to users
- B. (Easy) [Require MFA](#)

I. VPC

- A. (Easy) - Structure a set of subnets for each type of dataset (e.g. a set of subnets for RDS. One for Dynamo, etc.) in the same VPC as the EKS cluster
- B. (Medium) - Consider adopting the [VPC Terraform Module](#) for optimal configuration by default. While your handwritten terraform is good, there's lots more considerations that the module will add for free for you.
- C. (Easy) - Add VPC Endpoints for used AWS Services, i.e. DynamoDB, ECR, etc. in the EKS VPC
- D. (Medium) - Create a Subnet specifically for each client/customer with a nat gateway to allow traffic only through it allowing for non-public internet access.
- E. (Easy) - Use SSM instead of SSH to allow node level access into EC2 Instances

- F. (Medium) - Use EIPs attached to the LBs for ingress-nginx to allow for allow lists by customers.
 - 1. Install AWS LoadBalancer Controller
 - 2. Provision a EIPs matching number of subnets attached to the load balancer
 - 3. Use the [annotation to assign it to the LB](#)

EKS

I. Scaling

- A. (Medium) - Manage nodes and scaling via [Karpenter](#). Karpenter makes the management of EKS nodes very simple. The primary benefit of using Karpenter is allowing you to control the size of the cluster based on the number of pods requested. Karpenter also allows you to embrace the use of spot instances for significant cost savings. See Appendix A for more information on running Karpenter.
- B. (Easy) - Add [Node Problem Detector](#). EKS Nodes will have problems. Disk issues, memory or CPU corruption, NTP issues, etc. Node Problem Detector will report those up to the API and mark them as unhealthy either for your remediation or coupled with Karpenter the removal and provisioning of replacement nodes.
- C. (Easy) - Use only a single Managed Node Group. Currently there are multiple mNGs being used. These should be transitioned to a single mNG of a very small instance size solely for on-demand requirements (e.g. Karpenter).

II. Observability

- A. (Easy) - Install [Linkerd](#). Linkerd provides minimally invasive observability metrics and mTLS inside the cluster.
- B. (Easy) - All intra cluster service communication should use cluster DNS names. (e.g. service.namespace.cluster.svc.local)
- C. (Medium) - Install [GroundCover](#) for immediate and perpetual cluster visibility for all network calls, all relevant prometheus metrics, and immediate log aggregation using Loki.
- D. (Easy) - If more visibility is required, install Open Telemetry and ship to Ground Cover

- E. (Easy) - Deadman Snitch to ensure your website is up and available from StatusCake or alternative

III. Authentication

- A. (Medium) - Swap from managing aws-auth configmap to [Auth API](#). Managing the configmap by hand or via ArgoCD is error prone. Managing via API and Terraform allows for a cleaner implementation and is more sustainable and replicable.
- B. (Medium) - Use [TailScale](#) to create a VPN for private access.
- C. (Easy) - Swap to using [Google OIDC/Github OIDC for ArgoCD access](#)

IV. Cluster Management

- A. (Easy) - Install and use [ExternalDNS for R53 Management](#). Allows for DNS management via annotations on the services/ingress.
- B. (Easy) - Swap to [CertManager](#) for provisioning and renewing certificates.
 - 1. Allows for free public SSL certificate management
 - 2. Allows for [E2E TLS Encryption](#) without effort. Including from the LB to the pods
- C. (Medium) - [Meta Application pattern in ArgoCD](#). Allows you to declaratively define what applications need to be declared into the cluster and to manage the installation from the very beginning rather than manually managing it.
- D. (Easy) - Namespaces by environment not by service.
- E. (Hard) - Swap from using terraform to using [Crossplane](#) to manage AWS infrastructure from inside k8s. Specifically application infrastructure like S3 buckets or RDS instances.

V. Service Deployment

- A. (Easy) - Create a small helm chart for de-duplication of service resources
 - 1. Use: <https://github.com/helm/chart-releaser-action> to create and publish the helm chart to GHCR or to ECR
- B. (Easy) - Test HPA for scaling (set CPU requests to 5m and trigger)
- C. (Easy) - Configure a Kubernetes Service Account for each microservice deployed. Name of service account should match the name of the microservice

- D. (Medium) - Swap to [IAM Authentication for Database connections](#). This allows for short lived credentials and minimizes possible data leakage.
- E. (Easy) - Install IngressClass only once per cluster, not once per app.
- F. (Easy) - Port on pods/containers should always have a name and that name should always start with the protocol being used: name: http-web, http-api, http-metrics.

VI. Security

- A. (Medium) - Consider the use of network policies to control network access inside the cluster and prevent/allow certain services from talking to one another
- B. (Easy) - Manage Secrets via [ExternalSecrets](#). This allows for GitOps management of secrets definition while retaining the use of SecretsManager for encryption
- C. (Medium) - Add the AWS Guard Duty EKS add-on or OSS product like [Falco](#) for in cluster runtime security.

Continuous Integration & Developer Experience

VII. Continuous Integration

- A. (Easy) - Install [Renovate Bot](#). Like Dependabot but better. Covers more dependencies, including Helm Chart versioning and terraform versioning.
- B. (Easy) - Configure OIDC for [Github Actions into AWS](#) where needed. Useful for things like logging into ECR or into the cluster, pushing code to S3, etc.
- C. (Easy) - Pre-Commit Recommendations. Add the following types of linting.
 - 1. Dockerfile linting (Hadolint)
 - 2. Yaml linting
- D. (Easy) - Require PRs, without review, for all changes
- E. (Medium) - Database migration linting, e.g. [Squawk](#)
- F. (Medium-Hard) - Utilize [Contract testing](#) in lieu of E2E or Integration testing

VIII. Developer Experience



- A. (Easy) - Creating a [Template Repository](#) in github that contains the following will allow you to easily stand up new microservices that continually refer to best practices.
 - 1. Cookiecutter template of base microservice code
 - 2. Helm Chart reference and template values.yaml
 - 3. Easy templating script
- B. (Medium) - Consider the use for [devcontainers](#) and/or remote development environment in order to standardize tooling for developers and ease onboarding/machine swapping
- C. (Guidance) - Useful K8s Tooling for Local/CLI access
 - 1. kubectl/kubens
 - 2. [Kubectl aliases](#)


Continuous Deployment/Delivery & Production Engineering

IX. Continuous Deployment/Delivery

- A. (Medium) - Integrate Feature Flags into application to allow for the separation of code release from business release.
- B. (Easy) - Swap to [Automatic Sync](#) for ArgoCD Applications
- C. (Easy) - Use Spacelift or Terraform Cloud for CD of terraform code. Means when you modify terraform you'll get a plan as part of the PR and can apply directly without needing local credentials.

X. Production engineering

- A. (Medium) - Use [Flagger](#) specifically to manage canary releases of microservices. Flagger allows for gradual canary releases based on SLI metrics for measuring health. Start with easy metrics, like 5xxs or increased latency and then move to more specific business defined metrics like logins
- B. (Medium) - Determine SLOs and SLIs for each service and consider the engineering effort required to achieve them.
- C. (Easy - Hard) - Use a tool like [k6](#) for continuous testing. Start easy and ramp up. Run these tests continuously in both non-prod and prod. This allows you to create a continuous baseline of traffic for canary releases to measure release success. This is where Integration/E2E testing is helpful to think



about. Make these tests generic; testing API endpoints that are idempotent and ensure that they measure the API.

- D. (Guidance) - Don't configure a non-prod cluster. Run in the same cluster if needed. Utilize network policies to prevent services across namespaces from talking to one another.
- E. (Guidance) - Having a non-prod environment is perfectly acceptable however the CD pipeline should be: trunk -> non-prod -> prod. This allows you to confirm the functionality of your canary releases and general infrastructure.

Application Architecture

XI. Caching

- A. (Guidance) - [When to cache what](#). <- Article talks about useful considerations for local vs remote caches. Only extra guidance is to avoid ever caching authentication data.