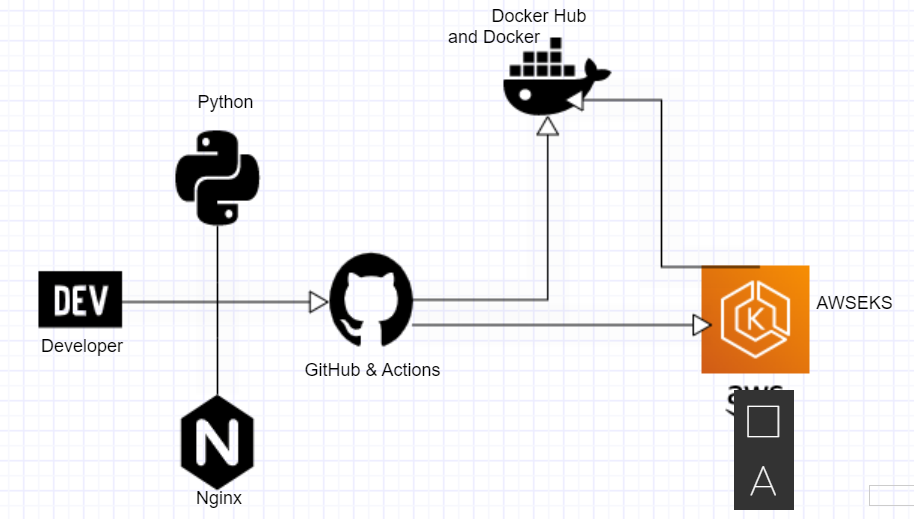
**POC Synch Project to achieve HA 99.999 %**



**Architecture of the app how it is achieved**

**Developer**: To develop code required for application functionality.

Might also include integration tests, UI tests, Load tests, Performance Tests and Unit

tests etc.

Finally, to push the code into any of the Source Code repo for ex: GITHUB.

**Python**: Programming used to develop the app that serves the business use case, can be any other language.

**Nginx**: Webserver which forwards requests from browser to the app.

**Wsgi:** Acts as interface which takes requests from webserver and sends it to appropriate python app.

**GitHub Actions**: Acts as a Continuous Integration or Continuous delivery runner to automate the deployment activities at the source code level.

**Docker**: Container to create the image for deploying the webapp, The image will be stored in docker repo.

**EKS**: Aws Kubernetes engine to run the docker image for achieving high availability.

**Terraform**: Infrastructure automation tool to build aws required services.

Diagram

Description automatically generated

Above diagram only represents having the k8 cluster deployed in multi-AZ in a single region, but a global business critical application is best if deployed using 3 az approach and on multiple region.

1. Monitoring: Using cloud watch to monitor AWS metrics. Prometheus and Grafana can be used for monitoring and viewing metrics from Nodes, Pods, Deployments, etc.
2. If Database is required Amazon RDS or Aurora both are exceptionally good as RDS grows its read replicas automatically when there is a need and Aurora can grow storage upto 64 GB with 10 GB increments.
3. Implementing changes using canary or blue green deployments making sure deployments are fully automated with rollback automation. Maintain Runbooks for rigorous reporting requirements and performance tracking. Also maintaining a playbook to define root cause if for any issue.
4. Backup data if it is database or related to ebs at regular intervals.
5. If possible, use three availability zone approach as its gives 50% peak static stability. If two AZs are used it might lead to 100% peak stability costing more.
6. Use Amazon cloud front for global caching.
7. Plan for DR either with Runbooks or using automation.

Spread worker nodes across more availability zones.

1. Use amazon global accelerator. Basically it acts as amazon own internet which optimizes the path and provides two static ips which routes traffic from users to our closes application end point.
2. Amazon Route 53 : DNS requests has to be handled by Amazon Route 53, a highly available Domain Name System (DNS) service.
3. Amazon CloudFront: is a content distribution network (CDN) with edge locations around the globe. It can cache static and streaming content and deliver dynamic content with low latency from locations close to the customers.
4. Amazon S3: To store static content, log files or videos if any and also store information from cloud front.
5. At heart we use EKS to deploy all the containers in PODS which will scale automatically in case of load or any pod failure.
6. DynamoDB a nosql database or an Amazon RDS or Aurora based on the requirement.
7. Elastic cache to store session data and can be used to retrieve without a I/O load to Dynamo DB.

NOTE:

1. You can connect applications hosted in Amazon EKS clusters in multiple AWS Regions over private network using [inter-region VPC peering](https://docs.aws.amazon.com/vpc/latest/peering/what-is-vpc-peering.html).
2. Gitops tools like flux2 and argocd can be used to connect to kube cluster apis and based on the state the change happened at git level can be automatically detected and applied. Used for multiple concurrent deployments.

CAP THEORY: consistency, availability and partition.

We need global app meaning we are doing partition and aiming for availability so we cannot achieve consistency.