# Traffic flip full automated deployment with Ansible

This project demonstrates the deployment of an Amazon EKS cluster and basic services like aws load balancer using Ansible and Amazon Cloud formation. The deployment is fully automated and will enable you to start learning and testing the elasticity and agility of AWS Services used with Kubernetes based microservice architectures.

This project is an example of an deployment and meant to be used for testing and learning purposes only.

**Friendly reminder < Please use the**[**AWS pricing calculator**](https://calculator.aws/#/estimate)**to an estimation beforehand>**

**Table of Contents**

1. [Getting started](https://github.com/aws-samples/aws-amazon-eks-ansible-example?tab=readme-ov-file#Getting-started)
2. [Prerequisites](https://github.com/aws-samples/aws-amazon-eks-ansible-example?tab=readme-ov-file#Prerequisites)
3. [Parameters](https://github.com/aws-samples/aws-amazon-eks-ansible-example?tab=readme-ov-file#Parameters)
4. [Templates](https://github.com/aws-samples/aws-amazon-eks-ansible-example?tab=readme-ov-file#Templates)
5. [Testing](https://github.com/aws-samples/aws-amazon-eks-ansible-example?tab=readme-ov-file#Testing)
6. Cloud architecture
7. [Resources](https://github.com/aws-samples/aws-amazon-eks-ansible-example?tab=readme-ov-file#Resources)
8. [Security](https://github.com/aws-samples/aws-amazon-eks-ansible-example?tab=readme-ov-file#Security)
9. [License](https://github.com/aws-samples/aws-amazon-eks-ansible-example?tab=readme-ov-file#License)

**Getting started**

Just a few steps are needed to get started with the example deployment. the deployment process is separated in a cluster deployment containing the creation of the Amazon EKS cluster itself and various cluster extensions and optional demo/example deployments to show the Amazon EKS cluster and the extensions in action.

You may use the [deployment playbook](https://github.com/aws-samples/aws-amazon-eks-ansible-example/blob/main/eks-deploy-cluster.playbook.yaml) for the [automatic deployment](https://github.com/aws-samples/aws-amazon-eks-ansible-example?tab=readme-ov-file#automatic) of Amazon EKS via Ansible. To login into the Amazon EKS deployment from your AWS Account .

**Prerequisites**

To run the Ansible based deployment you need to have some software installed and configured on your device:

* bash (zsh, csh, sh should also work, not tested though)
* an [installed and configured](https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-install.html)aws-cli
* [a named profile](https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-profiles.html) at the aws-cli configuration reflecting the account you are planning to use for the deployment
* [jq](https://stedolan.github.io/jq/)
* a working [Ansible installation](https://docs.ansible.com/ansible/latest/installation_guide/intro_installation.html)
* an install of the community kubernetes [Ansible module](https://github.com/ansible-collections/community.kubernetes)
* some pip modules: botocore boto boto3
* clone the following GitHub repo < [Green-Dot-Corporation/re-terraform (github.com)](https://github.com/Green-Dot-Corporation/re-terraform) >

**Explanation of the Ansible Playbook:**

Let’s take a closer look at the main contents of the `ansible1.yml` Ansible playbook:

1. Creating AWS Infrastructure:

The first play in the playbook is responsible for creating the required AWS infrastructure, including EC2 instances, security groups, and a key pair for secure access.

— region, project, and env are variables that can be customized based on your AWS region and project environment.

— The vars\_files section includes keys.yml, where you must provide your AWS access key and secret key.

— The playbook uses the amazon.aws.ec2\_key module to create a key pair and save it locally as project-env.pem.

[Note: Ensure that you have the necessary IAM permissions to create EC2 instances, security groups, and key pairs.]

2. Configuring Security Groups:

The second task configures the security group, allowing inbound traffic on ports 22 (SSH), 80 (HTTP), and 443 (HTTPS) to the instances.

3. Creating EC2 Instances:

The third task provisions EC2 instances using the amazon.aws.ec2\_instance module. It creates four t2.micro instances with the specified tags, security groups, and other configurations.

4. Gathering Instance Information:

The fourth task collects information about the newly created instances using the amazon.aws.ec2\_instance\_info module. This information will be used later to set up the dynamic inventory.

5. Creating Dynamic Inventory:

The fifth task creates a dynamic inventory by adding the instances to the zomato\_prod\_servers group, allowing Ansible to manage them.

6. Hosting on \_prod\_servers:

The second play is responsible for hosting the web application on the provisioned instances. It installs necessary packages, configures services, and copies files to the server.

**Parameters**

**Deployment parameters:**

Some static deployment variables are to be altered/placed into the [vars/static/definitions.yaml](https://github.com/aws-samples/aws-amazon-eks-ansible-example/blob/main/vars/static/definitions.yaml)

| **Parameter Name** | **Default Value** | **Description** | **Comment** |
| --- | --- | --- | --- |
| eksexample\_region | eu-central-1 | the region to be used for deployments | define the region to be used. Please be aware that the Amazon EKS on AWS Fargate used in some examples is not available in all [regions](https://docs.aws.amazon.com/eks/latest/userguide/fargate.html) |
| eksexample\_worker\_desiredcount | 2 | desired worker nodes to start with |  |
| eksexample\_worker\_maxcount | 10 | maximum workers to be provisoned if scaling out the cluster |  |
| eksexample\_worker\_mincount | 2 | minimum worker nodes | i recommend at least 2 |
| eksexample\_worker\_instancetype | t3a.medium | instance size of the worker nodes |  |
| eksexample\_bastion\_instancetype | t3a.small | instance size of the bastion host |  |
| eksexample\_clustername | ansible-eks-testcluster | name of the Amazon EKS cluster |  |
| eksexample\_clusterversion | 1.18 | version of the Amazon EKS cluster | versions <1.16 are not tested with this automation |
| eksexample\_aws\_profilename | ansible | the [profile name setup](https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-profiles.html) for the local awscli | i recommend to setup a local profile. if you decide to shift it directly to instance\_profile based EC2 admin instances, alter the ansible module parameters to not use "profile:" |

rename the [vars/static/custom\_definitions.yaml](https://github.com/aws-samples/aws-amazon-eks-ansible-example/blob/main/vars/static/custom_definitions.yaml.example) and alter the parameters according to your needs

| **Parameter Name** | **Default Value** | **Description** | **Comment** |
| --- | --- | --- | --- |
| eksexample\_hostedzoneid |  | the ID of your Route53 Zone where the DNS automation should work on. |  |
| eksexample\_hostedzonename |  | the Domainname of the Hosted Zone on Route53 |  |

**Deployment**

The deployment will take approx 5 minutes.

**Template structure and deployment workflow**

The Deployment consists of one main playbook triggering multible tasks, cloud formation templates .

**Playbooks**

* eks-deploy-cluster.playbook.yaml: this playbook starts the overall deployment of the Amazon EKS cluster and triggers also the deployment of all extensions. can be started with ansible-playbook ./eks-deploy-cluster.playbook.yaml
* eks-destroy-cluster.playbook.yaml: this playbook destroys the whole deployment. If you deployed the example deployments into the cluster, make sure these get destroyed first using the ./docs/examples/destroy-examples.playbook.yaml playbook.
* ./docs/examples/deploy-examples.playbook.yaml: this playbook will deploy some microservice and overall deployment examples to demonstrate the functionality of the extensions.
* ./docs/examples/destroy-examples.playbook.yaml: will remove the example deployments from the cluster (but leaves the cluster intact)

**Tasks**

* eks-cluster-autoscaler.task.yaml: setup of the [cluster-autoscaler](https://github.com/kubernetes/autoscaler/tree/master/cluster-autoscaler)
* eks-container-insights.task.yaml: enable [container insights](https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/ContainerInsights.html) for the Amazon EKS cluster
* eks-ingress-controller.task.yaml: setup of the [aws-load-balancer-controller](https://github.com/kubernetes-sigs/aws-load-balancer-controller) to automate service exposure

**Cloud formation Templates**

* eks-bastion.template.yaml
* eks-cluster-autoscaler-iam.template.yaml: provisioning of the IAM Policy granting access for the cluster autoscaler to Amazon EC2 and EC2 Autoscaling groups.
* eks-container-insights-iam.template.yaml: provisioning of the IAM Policy allowing Amazon Cloudwatch Access via the Worker Nodes
* eks-external-dns-iam.template.yaml: provisioning of the IAM Policy granting access for the external-dns pods to Route53
* eks-ingress-controller-iam.template.yaml: provisioning of the IAM Policy granting access for the aws-load-balancer-controller towards Elastic Load Balancing
* eks-storage-provider-ebscsi-iam.template.yaml: IAM Policies to Allow EBS Access via the CSI Driver Deployment
* eks-storage-provider-efscsi-storage.template.yaml: provisioning of the EFS FileSystem, Mountpoints and related Securitygroups

**Testing**

A screen shot of a computer code

Description automatically generated

A screenshot of a computer program

Description automatically generated

**Cloud architecture diagram**

A computer screen shot of a diagram

Description automatically generated