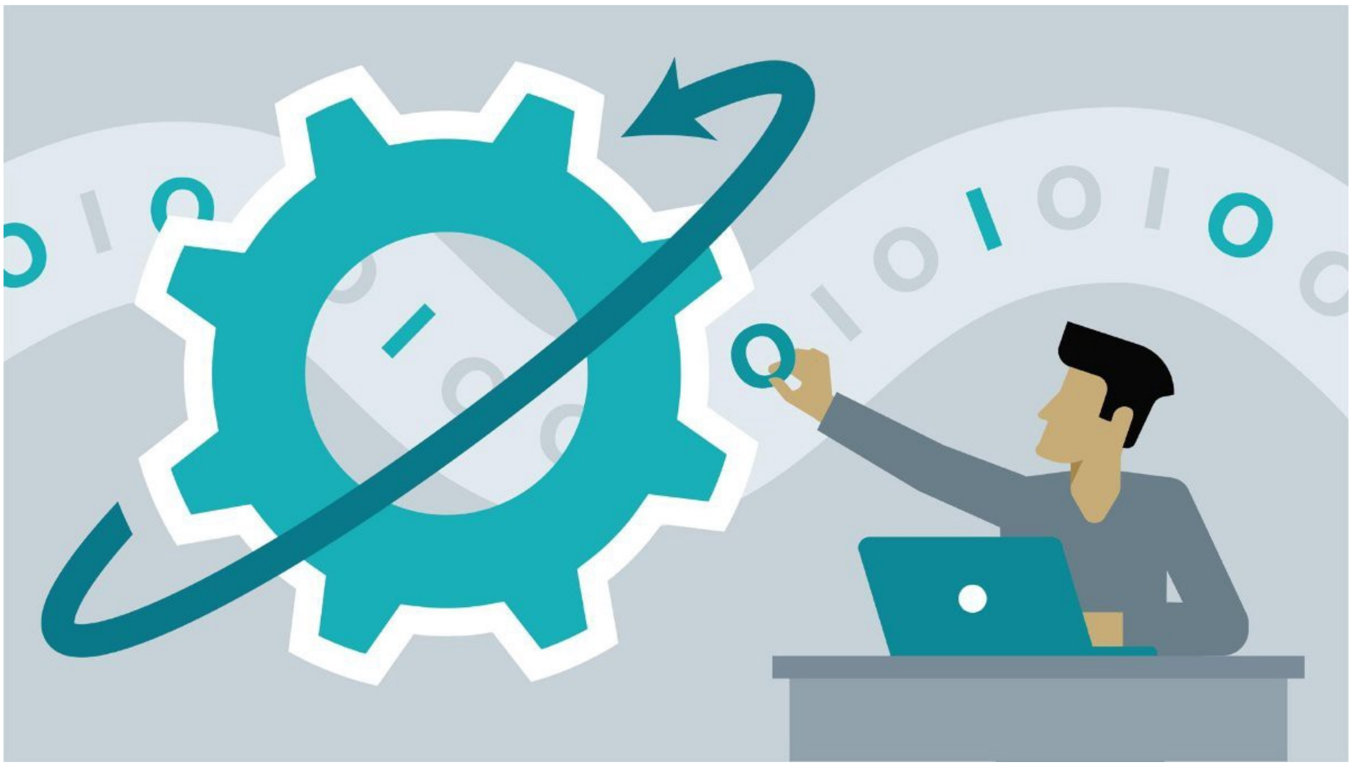
How to setup a free CI/CD with GoCD using Docker, managed with terraform

Craig Godden-Payne
Oct 3, 2018 · 6 min read ★

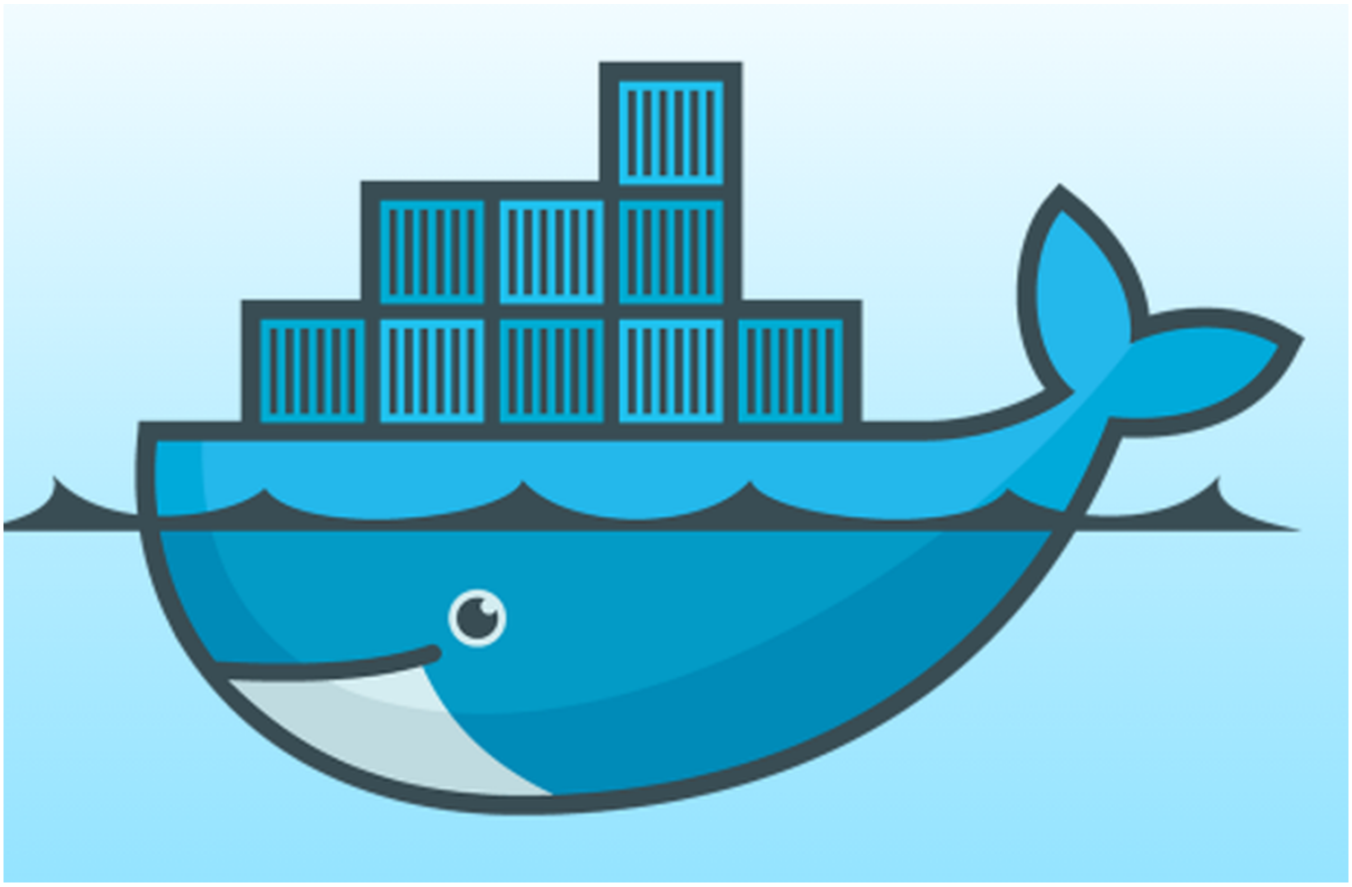


Today we're going to setup a CI server in a way that is easy to manage and scale. A natural fit for this, would be to use docker as we can define the software once, and run many containers.

The cloud service provider we are going to host in, is AWS. We are also going to be hosting within Elastic Container Service (ECS). We'll store images in ECR — Elastic

Container Repository and we'll use terraform, so that changes to the overall infrastructure can be managed through a deployment.

Let's begin.



. . .

Overview of what we are doing

There are 4 stages to the deployment of what we are going to achieve.

- Create the Container Host
- Create the Elastic Container Register
- Build and Upload the Docker Image
- Create the Elastic Container Service

These are relatively straightforward to setup in terraform and bash, and can be tailored to your own requirements.

Setting up the container host

Setting up the container host in terraform is pretty straight forward, as it is just an EC2 instance, running an optimised ECS AMI. You can opt to use the Amazon ecsInstanceRole for the IAM role, rather than create your own, as it already exists in AWS.

The container host will house different GoCD agents.

You can check out the documentation here: IAM docs for ecsInstanceRole

```
resource "aws_ecs_cluster" "ci-container-cluster" {
  name = "CI-Container-Cluster"
}

resource "aws_instance" "ci-container-host-1" {
  ami           = "ami-2e9866c5" #ecs optimized image
  instance_type = "t2.medium"

  vpc_security_group_ids      = ["${aws_security_group.ci-container-
host-security-group.id}"]
  subnet_id                   =
"${element(data.aws_subnet_ids.public_subnets.ids, 0)}"
  key_name                    = "infrastructure"
  associate_public_ip_address = true

  user_data = <<EOF
#!/bin/bash
echo ECS_CLUSTER=${aws_ecs_cluster.ci-container-cluster.name} >>
/etc/ecs/ecs.config
echo ECS_BACKEND_HOST= >> /etc/ecs/ecs.config
echo NO_PROXY=169.254.169.254,169.254.170.2,/var/run/docker.sock >>
/etc/ecs/ecs.config
echo 'vm.max_map_count = 262144' >> /etc/sysctl.conf
sysctl -p
EOF

  iam_instance_profile = "${aws_iam_instance_profile.ingest.name}"

  tags {
    Name = "CI-Container-Host-1"
  }
}
```

```
data "aws_iam_role" "ecsInstanceRole" {
  name = "ecsInstanceRole"
}
resource "aws_iam_instance_profile" "ingest" {
  name  = "ingest_profile"
  role = "${data.aws_iam_role.ecsInstanceRole.name}"
}
```

Setting up the Elastic Container Registry

You can create a terraform module for this, as it is a very repeating process if you create many different agent types.

You can create different agent types by reusing modules, like this.

```
#container.tf

module "gocd-agent-git-repository" {
  source     = "../modules/ci-repository"
  agent_name = "gocd-agent-git"
}

module "gocd-agent-terraform-repository" {
  source     = "../modules/ci-repository"
  agent_name = "gocd-agent-terraform"
}

module "gocd-agent-node-repository" {
  source     = "../modules/ci-repository"
  agent_name = "gocd-agent-node"
}

module "gocd-agent-dotnetcore-repository" {
  source     = "../modules/ci-repository"
  agent_name = "gocd-agent-dotnetcore"
}

module "gocd-agent-docker-repository" {
  source     = "../modules/ci-repository"
  agent_name = "gocd-agent-docker"
}
```

Here I specified 5 different repositories, for each agent type:

- git
- terraform

- node
- dotnetcore
- docker (docker in docker)

When using the module, it makes the terraform straightforward, as it will look like this:

```
#ecr module

variable "agent_name" {}

resource "aws_ecr_repository" "agent-repository" {
  name = "${var.agent_name}"
}

resource "aws_ecr_repository_policy" "ci-agent-ecr-policy" {
  repository = "${aws_ecr_repository.agent-repository.name}"

  policy = <<EOF
{
    "Version": "2008-10-17",
    "Statement": [
        {
            "Sid": "new policy",
            "Effect": "Allow",
            "Principal": "*",
            "Action": [
                "ecr:GetDownloadUrlForLayer",
                "ecr:BatchGetImage",
                "ecr:BatchCheckLayerAvailability",
                "ecr:PutImage",
                "ecr:InitiateLayerUpload",
                "ecr:UploadLayerPart",
                "ecr:CompleteLayerUpload",
                "ecr:DescribeRepositories",
                "ecr:GetRepositoryPolicy",
                "ecr:ListImages",
                "ecr:DeleteRepository",
                "ecr:BatchDeleteImage",
                "ecr:SetRepositoryPolicy",
                "ecr:DeleteRepositoryPolicy"
            ]
        }
    ]
}
EOF
}

resource "aws_ecr_lifecycle_policy" "ci-agent-ecr-policy" {
  repository = "${aws_ecr_repository.agent-repository.name}"
  policy = <<EOF
{
    "rules": [
        {
            "rulePriority": 1,
            "description": "Keep last 5 images",
            "selection": {
                "tagStatus": "tagged",
                "tagPrefixList": ["v"],
                "countType": "imageCountMoreThan",
                "countNumber": 5
            },
            "action": {
                "type": "expire"
            }
        }
    ]
}
EOF
}
```

This will now give us something to upload our docker image to.

Building and Uploading the Docker Image.

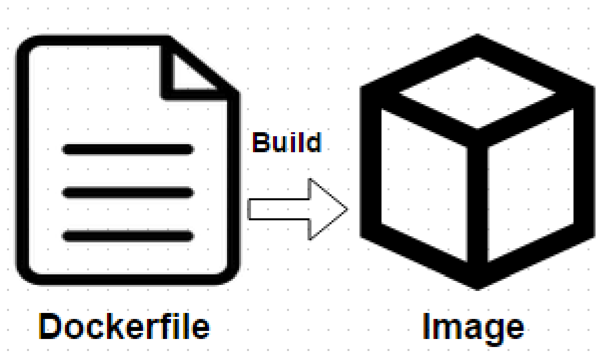
You can store each of the agent dockerfiles within a directory, with the name of the agent as the name of the agent, then use a simple bash script to loop through each of the dockerfiles, build it and upload the the elastic container registry

```
#build.sh
#!/bin/bash
set -e

for dir in `find . -type d`
do

echo "using directory "$dir
if [ $dir = "." ]; then
    echo ""
else

    BASE_REPO=XXXXXXXXXXXX.dkr.ecr.eu-west-2.amazonaws.com
    IMAGE_NAME=${dir:2}
    VERSION_LATEST=latest
    VERSION=2.0
    echo "ImageName:"$IMAGE_NAME
    eval $(aws ecr get-login --region eu-west-2 --no-include-email)
    sleep 1

    docker build $dir -t $IMAGE_NAME:$VERSION
    docker tag $IMAGE_NAME:$VERSION
```

```
$BASE_REPO/$IMAGE_NAME:$VERSION_LATEST
    docker tag $IMAGE_NAME:$VERSION $BASE_REPO/$IMAGE_NAME:$VERSION
    docker push $BASE_REPO/$IMAGE_NAME:$VERSION
    docker push $BASE_REPO/$IMAGE_NAME:$VERSION_LATEST

fi

done
```

Here is an example of one of the dockerfiles we created. This one is for the agent that runs terraform files

```
FROM gocd/gocd-agent-ubuntu-16.04:v18.7.0
RUN apt-get update -y && apt-get upgrade -y && \
    apt-get install -y bash tree tar zip unzip xz-utils

RUN curl -o terraform.zip
https://releases.hashicorp.com/terraform/0.11.1/terraform_0.11.1_lin
ux_amd64.zip && \
    unzip terraform.zip && \
    mv terraform /usr/local/bin/

RUN apt-get install -y python-setuptools python-dev build-essential
&& \
    easy_install pip && \
    pip install --upgrade pip && \
    pip install awscli

ENV GO_SERVER_URL=https://XXXX.XXXXXXXX.com:8154/go/
ENV AGENT_AUTO_REGISTER_KEY=XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX
ENV AGENT_AUTO_REGISTER_RESOURCES=terraform,aws-cli
ENV AGENT_AUTO_REGISTER_ENVIRONMENTS=Build,Infrastructure,Prod,QA
ENV AGENT_AUTO_REGISTER_HOSTNAME=gocd-agent-terraform

ENTRYPOINT /docker-entrypoint.sh
```



. . .

Setting up the Elastic Container Service

The elastic container service starts docker images, and manages them on the container host.

You can use config similar to this to create a service, most of the infra is so similar, that again it made sense to move this into a module.

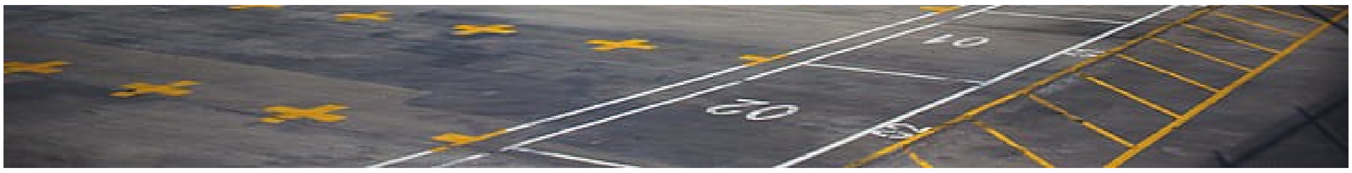
```
#containers.tf
module "gocd-agent-git-container" {
  source              = "../modules/ci-agent"
  agent_name          = "gocd-agent-git"
  ecr_base_repository = "${var.repository}"
  tag                 = "2.0"
  memory_reservation  = 128
  instance_count      = 2
}

module "gocd-agent-terraform-container" {
  source              = "../modules/ci-agent"
  agent_name          = "gocd-agent-terraform"
  ecr_base_repository = "${var.repository}"
  tag                 = "2.0"
  memory_reservation  = 128
  instance_count      = 2
}

module "gocd-agent-node-container" {
  source              = "../modules/ci-agent"
  agent_name          = "gocd-agent-node"
  ecr_base_repository = "${var.repository}"
  tag                 = "2.0"
  memory_reservation  = 384
  instance_count      = 2
}
module "gocd-agent-dotnetcore-container" {
  source              = "../modules/ci-agent"
  agent_name          = "gocd-agent-dotnetcore"
  ecr_base_repository = "${var.repository}"
  tag                 = "2.0"
  memory_reservation  = 384
  instance_count      = 1
}

module "gocd-agent-docker-container" {
  source              = "../modules/ci-agent"
  agent_name          = "gocd-agent-docker"
  ecr_base_repository = "${var.repository}"
  tag                 = "2.0"
  memory_reservation  = 256
  instance_count      = 1
}
```

The in the actual definition for the module:

```
#container-module.tf

variable "instance_count" {
  default = 1
}
variable "agent_name" {}

variable "tag" {
  default = "latest"
}

variable "memory_reservation" {
  default = 256
}

variable "ecr_base_repository" {}

resource "aws_ecs_service" "ci-agent-service" {
  name                = "${var.agent_name}"
  cluster             = "${data.aws_ecs_cluster.ci-cluster.id}"
  task_definition     = "${aws_ecs_task_definition.agent-
definition.arn}"
  desired_count       = "${var.instance_count}"
  scheduling_strategy = "REPLICA"
}

resource "aws_ecs_task_definition" "agent-definition" {
  family       = "${var.agent_name}"
  network_mode = "host"
  volume = {
    name      = "dockerdaemon"
    host_path = "/var/run/docker.sock"
  }
  container_definitions = <<DEFINITION
[
  {
    "name": "${var.agent_name}",
    "image":
"${var.ecr_base_repository}/${var.agent_name}:${var.tag}",
    "hostname": "${var.agent_name}",
    "essential": true,
    "privileged": true,
    "memoryReservation": ${var.memory_reservation},
    "mountPoints": [
        {
          "sourceVolume": "dockerdaemon",
          "containerPath": "/var/run/docker.sock"
        }
    ],
    "requiresAttributes": [
        {
        "value": null,
        "name": "com.amazonaws.ecs.capability.ecr-auth",
        "targetId": null,
        "targetType": null
        },
        {
        "value": null,
        "name": "com.amazonaws.ecs.capability.task-iam-role",
        "targetId": null,
        "targetType": null
        },
        {
        "value": null,
        "name": "com.amazonaws.ecs.capability.docker-remote-
api.1.19",
        "targetId": null,
        "targetType": null
        }
    ]
  }
]
DEFINITION
}

data "aws_ecs_cluster" "ci-cluster" {
  cluster_name = "CI-Container-Cluster"
}
```

The terraform for this again is quite straightforward, its probably worth that:

- We are mounting /var/run/docker.sock from the container to the host, so that rather than running docker in docker on the docker agent, we are utilising the host docker service, which prevents many issues -



Docker Gocd DevOps Terraform Continuous Integration