Docker & AWS

Fabrizio Ciacchi

Explained simple for PHP Developers :)

Who am I?

Fabrizio Ciacchi / @ciacchi / ciacchi.it

Lead Software Developer @ Greta & Starks Apps GmbH.

Worked previously in Rocket Internet and Spryker Systems

PHP Developer from more than 10 years

I’m working with Symfony from version 2+

I want to tell a story...

... how I started the migration of Greta App:

- Legacy Application on Hosted Server

I have been requested to:

- Create the whole AWS deployment system
- Migrate to a new Symfony Application
- Ensure security and best practices

Step One / Clean-up

- Clean-up your passwords and conf (use .env)
- Clean-up your data - you should have ‘demo’ data for your dev and staging machines. It also facilitate tests and development
  - Put DB and Content in different repositories
- Update your application:
  - Latest version of PHP, MySQL and other softwares
  - Latest version of your framework
  - If you use it, check and update your composer
- Move your cache and tmp files to Redis (or other storage)

Step two / Dockerize

- Very simple:
  - A virtual machine with a configuration file
- Why we use it?
  - Faster - it depends from the host system
  - Configuration based - it removes a lot of Admin/DevOPs issues
  - Reproducible environment
- What to use:
  - Bitnami Nginx
  - Bitnami PHP
  - Official MySQL
  - Bitnami has a lot of packages (Laravel, Redis, etc)
  - And... nibirrayy/docker-smtp

Step three / Still Docker

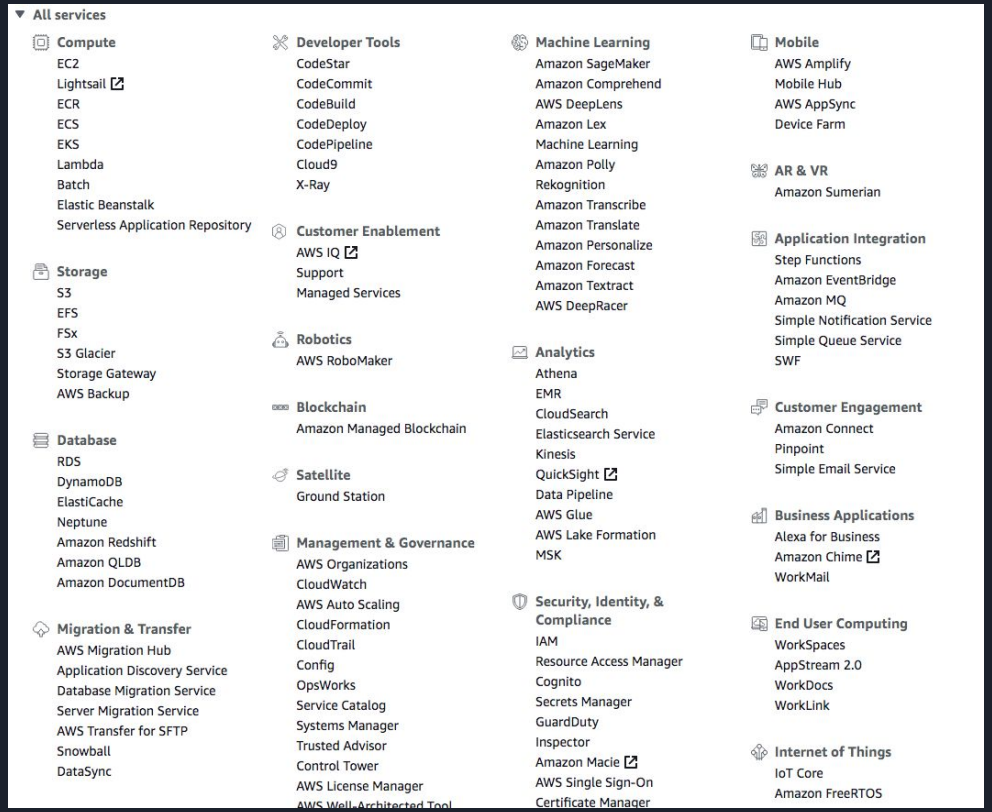
Why those packages?

- Bitnami packages are highly configurable
- MySQL package allows to import SQL files

Also remember:

- Use Docker-hub
- port mapping is to expose ports outside
- You'll probably need an SMTP server
- Use mounted volumes (ie: composer)
- Avoid custom Docker images (you'll have to compile your own image)

AWS



- So many services!!!
- Don’t get overwhelmed
- You might need a CC (ok you have also free tiers)

EC2 - your ‘machine’

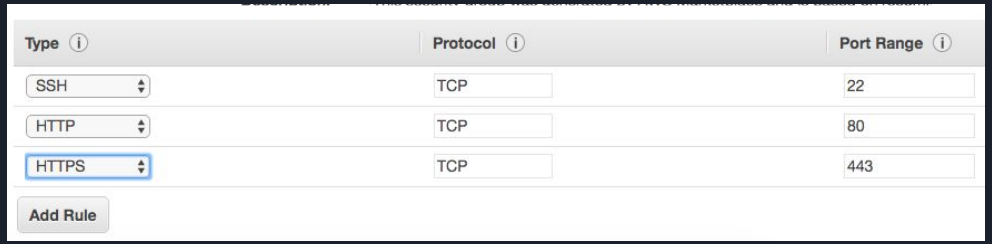
The first thing to do is to set-up your Staging machine.

Why Staging?

- First you want to start simple
- Also can be used as ‘base’ for Production

	Family	Type	vCPUs	Memory (GiB)
☐	General purpose	t2.nano	1	0.5
☐	General purpose	t2.micro Free tier eligible	1	1
☐	General purpose	t2.small	1	2
☑	General purpose	t2.medium	2	4

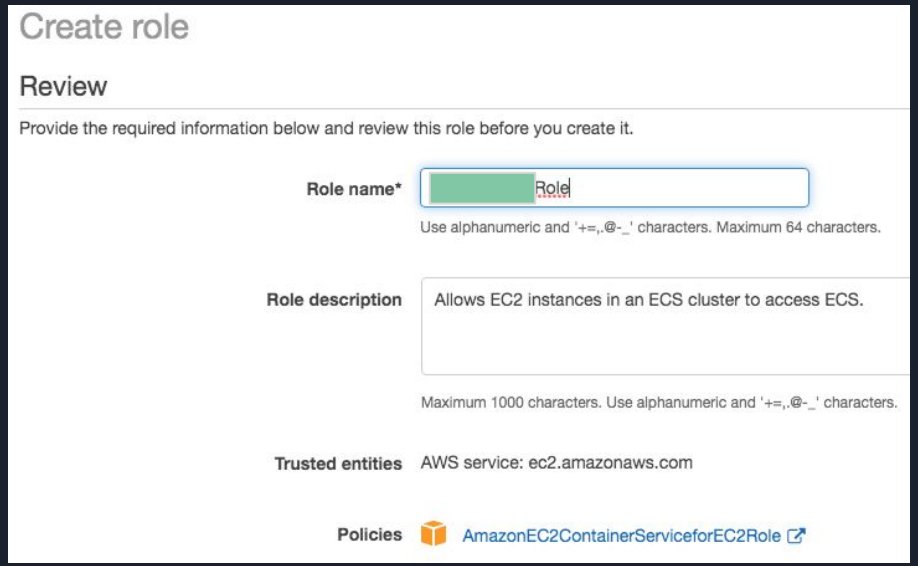
Configure Security Groups & Roles



Additionally you might want the IAM role

AmazonS3FullAccess

At the end you need to generate and download a PEM certificate!



Set-up Application + S3 Bucket

Clone your repository(-ies) in /home/ec2-user (do not store GIT credentials)

But sym-link them under ie: /opt/app

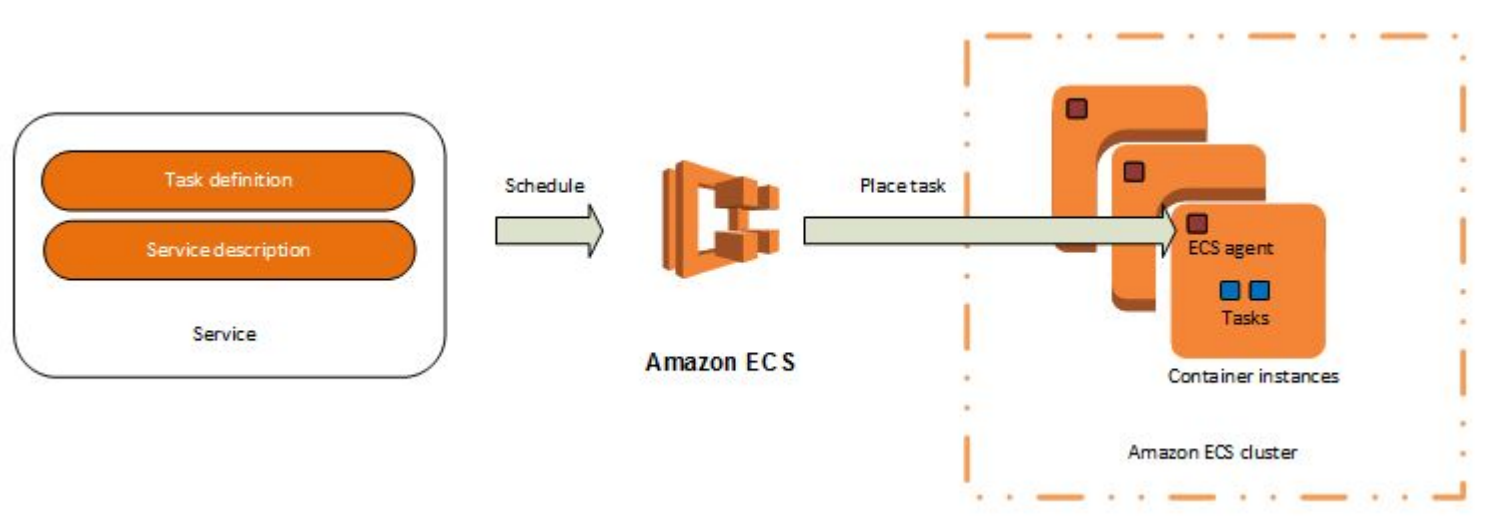
S3 Bucket is nearly infinite storage:

- 1 for Deployments
- 1 for Logging
- 1 for Storage (media, etc.)

User s3fs-fuse to mount the S3 bucket(s) in your EC2:
https://github.com/s3fs-fuse/s3fs-fuse.git

Re-mount at reboot with crontab (-e)

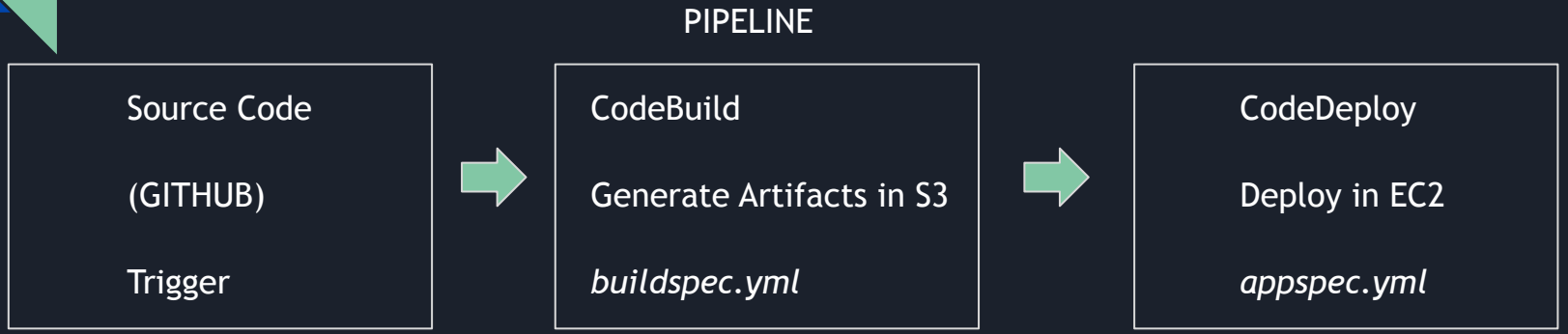
ECS - visually



ECS - Elastic Container Service

- Task is a mapping of your Docker file
  - Create the shared mounts (your folders in EC2)
  - Set-up the min (and max) for memory and CPU unit
  - Network ‘bridge’
  - Set-up services as non-essential (DB essential)
  - Add networking links so services can ‘see’ each other
  - Port mapping
- Create a Cluster and a Service associated with the Task.
- Install the ECS Agent in your EC2: https://docs.aws.amazon.com/AmazonECS/latest/developerguide/ecs-agent-install.html
- In etc/ecs/ecs.config set-up your ECS service name ECS_CLUSTER=staging

CodePipeline



- Define IAM Roles
- Install the Agent: https://docs.aws.amazon.com/codedeploy/latest/userguide/codedeploy-agent-operations-install.html
- Link:https://hackernoon.com/continuous-deployment-with-aws-codedeploy-github-d1eb97550b82
- Hooks:https://docs.aws.amazon.com/codedeploy/latest/userguide/reference-appspec-file-structure-hooks.html#appspec-hooks-ecs

Ok, but where is my website visible

Your EC2 instance has a public and private IP address. But that's not good (not secure)

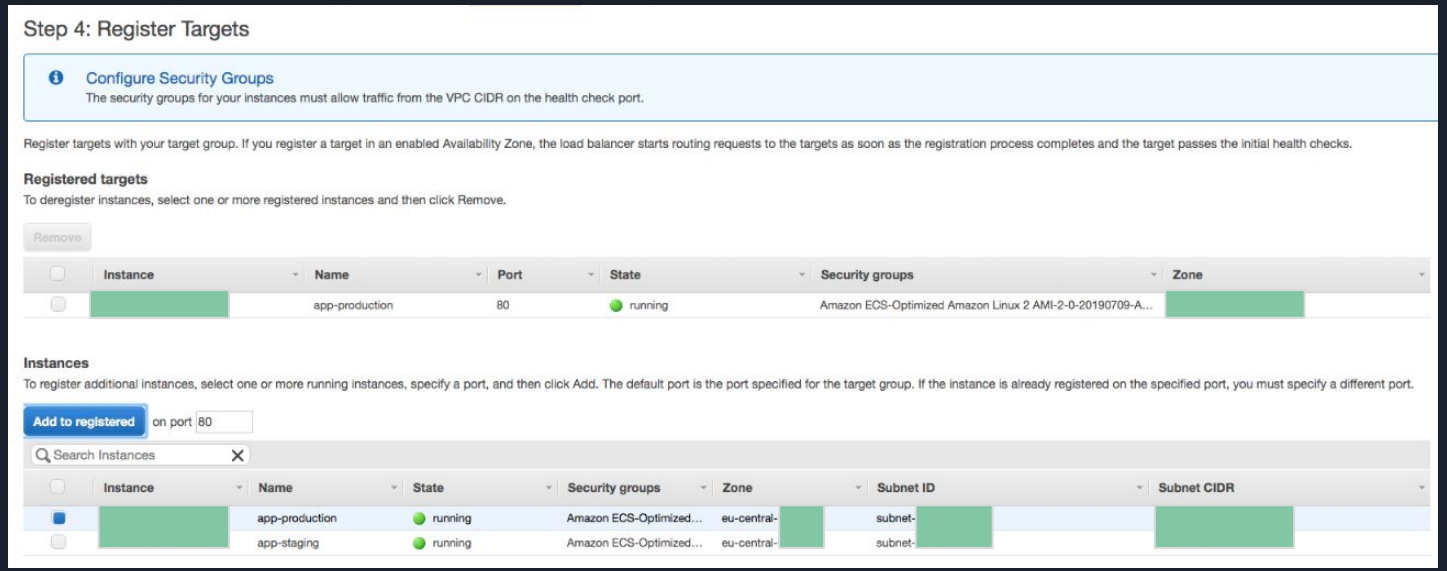
- Register or transfer your domain with Route53
- Set-up an Elastic IP address (public) for your staging instance
- Point staging.myapp.com to the Elastic IP
- Allow only HTTP/HTTPS to the public IP (no SSH, only private)
- Add your SSL certificate with AWS certificate manager

What about Production?

- Create a snapshot of Staging
- Create a new EC2 instance from it
  - Change the attribute name to production
  - Download the new PEM certificate
  - Log-in inside and change the data source (DB?), env files, and ECS cluster name
- Shut it down - do not terminate (it will destroy)
- Create Launch Template from it

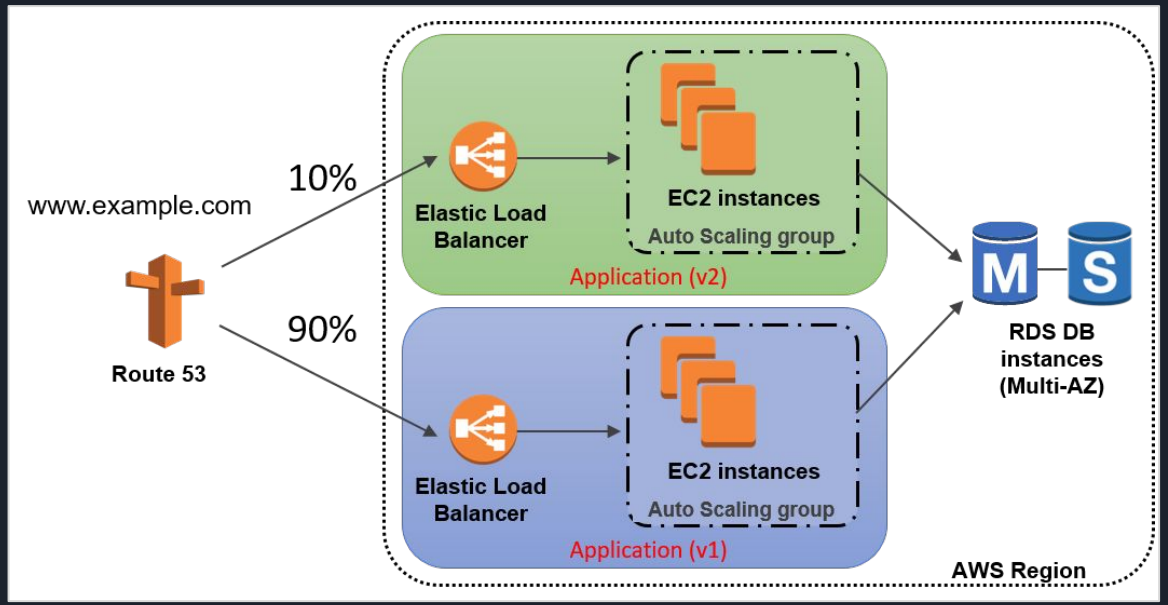
And then and then and then...

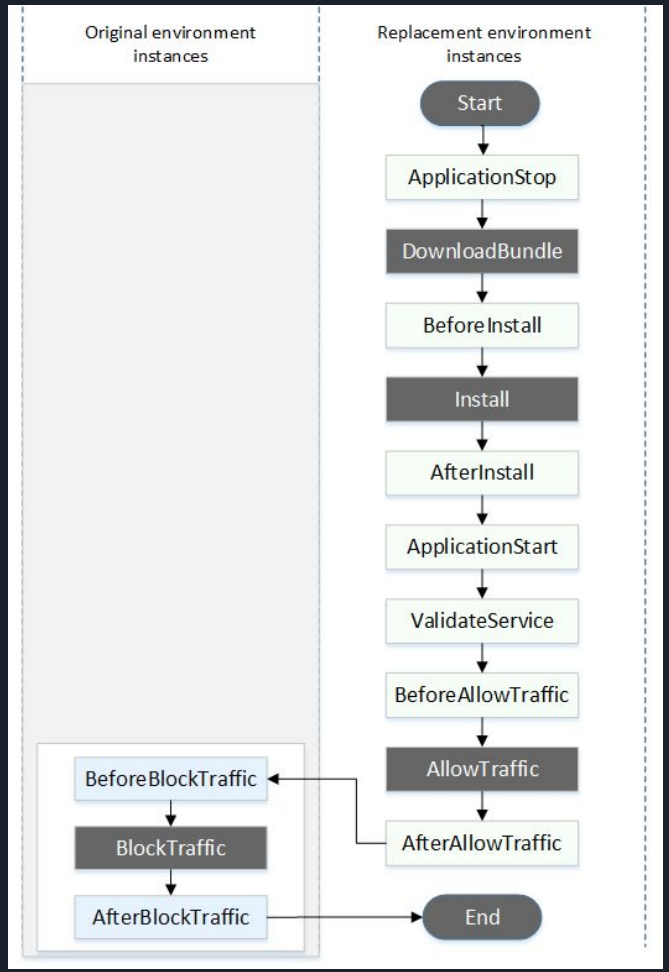
- Load balancer
- Target Group
- Auto-scaling + Launch Template
- Assign Elastic IP to load balancer
- Route53 to assign production to that IP (www and without)



CodePipeline

- Create a new ECS task + service
- Create a Blue/Green deployment
- Take care of the appspec hooks (now with routing traffic become more important)





Todo:

- Monitoring
- TravisCI
- Kibana

Thanks everybody :)

@ciacchi / ciacchi.it