Use Case: Running Docker Containers

1. Introduction

A new trend in software development is to break down applications into microservices and deploy them using containers. Containerization has many benefits over the traditional approach of using Virtual Machines. When adopting a microservices strategy, using containers is often the only way to success.

The benefits of running containers are (but are not limited to):

- Faster bootup (seconds instead of minutes with just virtualization)
- Better Continuous Integration (repeatable binary builds)
- Closer dev-prod parity (same binary for dev, qa and prod)
- Immutable infrastructure (container images can't be changed)
- Improved scaling (scale on a per container basis)

To make this architecture design work, containers must be orchestrated within a cluster of machines. When using AWS, EC2 Container Service (ECS) can be used to manage the lifecycle of the docker containers. The EC2 Container Registry (ECR), another AWS service, is used to store the images of the containers.

ECS can take care of starting containers from ECR and run the desired amount of containers. An Elastic Load Balancer (ELB) can be added for every microservice that needs to be deployed and that exposes some API or frontend to the customer or other microservices.

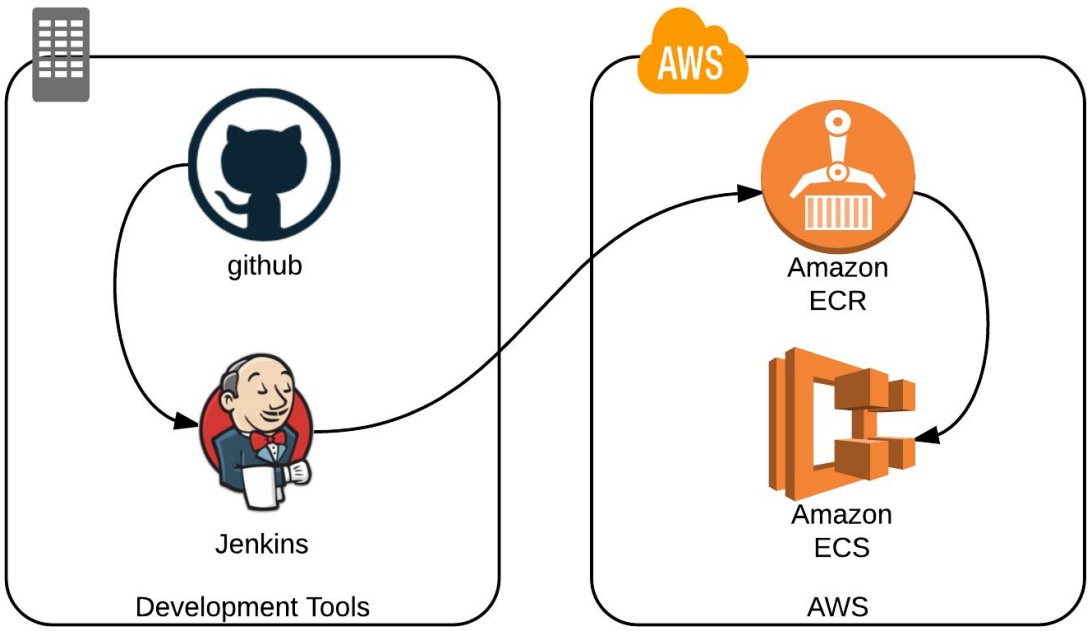
To achieve auto-discovery of microservices, consul can be added to the mix. A consul cluster consists of 3 or 5 nodes, which gives you a consistent, distributed key value store. By running an agent, all new microservices can be registered automatically to the consul cluster. A separate load balancer can be set-up to automatically detect new services in consul. At this point, when deploying a new service on the ECS cluster, it becomes immediately available through the load balancer on its own endpoint.

2. Architecture Diagram

2.1 The Delivery Pipeline

The delivery pipeline enables the developer to focus on what's really important: the app. Any change to any of the microservices can be picked up by Jenkins. Jenkins then triggers the rebuild of the Docker image. If the rebuild succeeds, the image can be pushed to Amazon ECR. To deploy the new changes on the dev / qa / production environment, Jenkins can change the ECS service definitions to make the new microservice available.

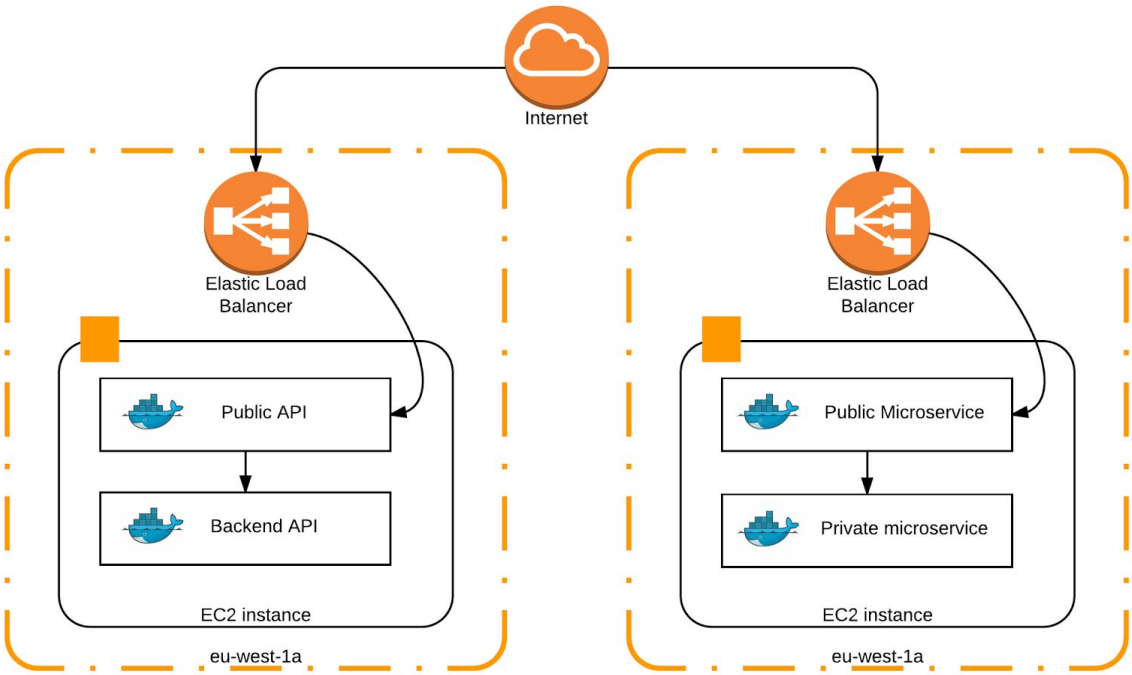
If the newly deploy application would not succeed its health checks, ECS will automatically roll back the older version.



2.2 Publicly facing microservices

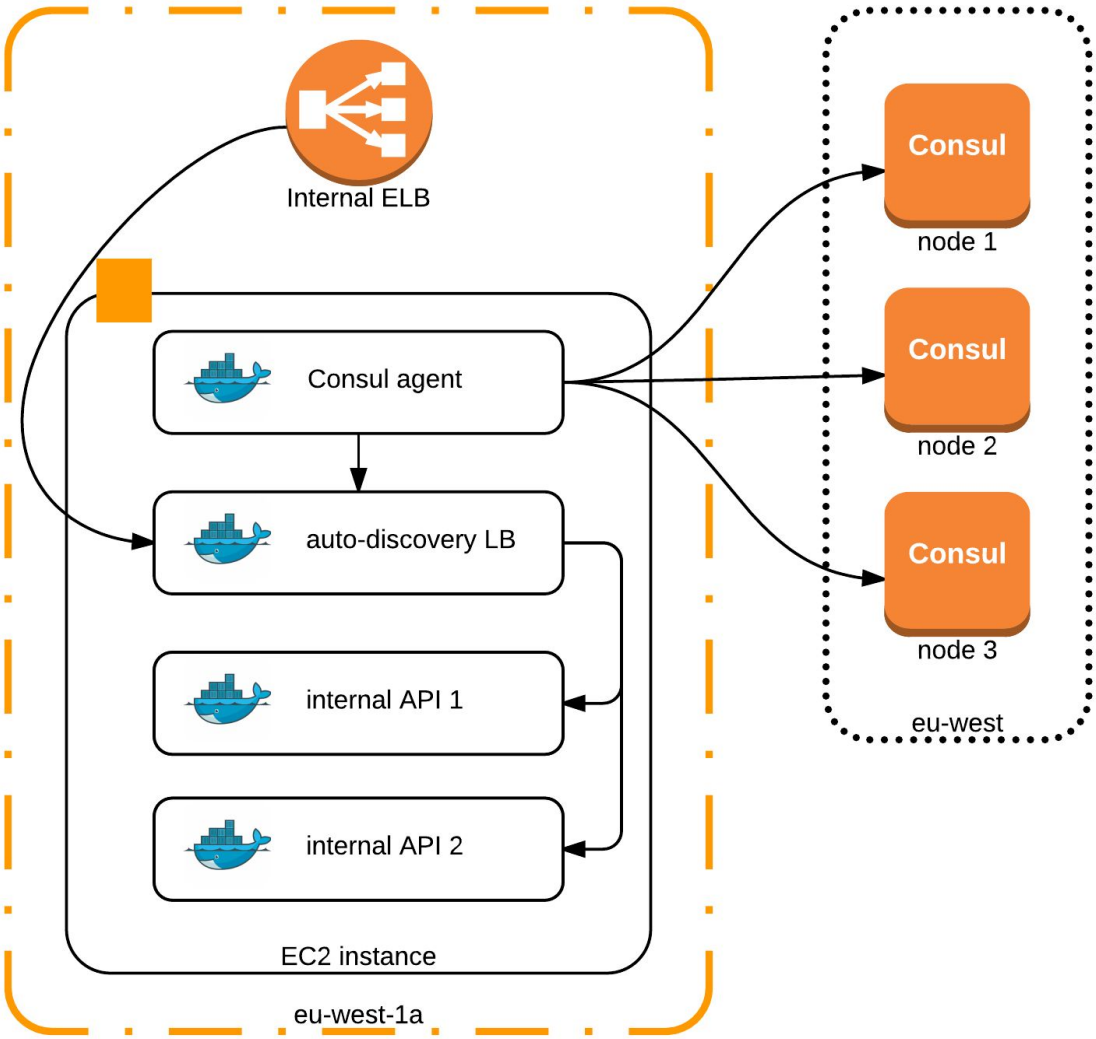
Publicly facing Docker containers can be made reachable using the standard Elastic Load Balancer from AWS. The load balancer checks the health of the containers in different availability zones (different datacenters in AWS). If one of the containers would become unhealthy (e.g. datacenter failure), the load balancer will redirect all traffic to the healthy container.

Any number of docker containers can be added to EC2 instances. More EC2 instances can be added to allow more container capacity in a certain availability zone. Typically 2 availability zones are used to provide high availability.



2.3 Internal facing microservices

Microservices can also be auto-discovered by adding a service called consul. When auto-discovery is enabled, after deploying a new microservice, the API becomes immediately available as an HTTP endpoint, e.g. api1.company.internal, without any additional configuration. This gives developers the capability to deploy any microservice they want, without any operational effort. This is one of the most wanted advantages of the micro-services architecture.



3. Setup

Infrastructure setup

The infrastructure is set up by in4it. After gathering the customers requirements, the infrastructure is provisioned using the customer's AWS credentials. The infrastructure blueprint (documentation) is issued and any relevant credentials are handed over to the customer.

Typically dev and optionally QA/Test are setup first, so the customer can prepare the application migration. Typical migration timelines go from a couple of weeks to months.

Application setup

To deploy the application, a Dockerfile needs to be added to the root of the project. This Dockerfile describes:

- The image to be used (e.g. nodejs, php-apache, redis)
- The ports to be exposed (a container port)
- Any volumes to be used (volumes can provide persistency for stateful applications, like databases)
- The command to start the application (e.g. npm start, apachectl start)
- Environment variables to pass to the application

Runtime configuration

The configuration variables can be set in different ways. They can either be set using environment variables, configuration files, or they can be stored in the distributed key/value store Consul. The latter is the recommended way when deploying microservices. The application should be modified to read key/value pairs from the consul cluster when the application starts. Different values can be stored depending on whether the application is in dev / qa / prod. Consul can be contacted using a simple HTTP API and also exposes DNS.

DevOps

The architecture allows for a DevOps approach:

- A git commit to develop branch triggers a build workflow in Jenkins (or any CI tool)
    - The application can be automatically compiled (if applicable)
    - The application can run its tests: unit tests, regression tests, etc
    - If all tests succeed, the docker container builds and gets pushed to ECR
    - Once the container is available in the repository, the deployment can be triggered
- Zero-downtime deployments are included in the standard production build
- Green/Blue deployments can be enabled, to make sure new deployments are healthy before they come live
- The deployment tools allow for a very short deployment cycle (multiple deployments per day are possible)

4. Support & Maintenance

Package:

Basic Support package	Max 1 codebase, max 1 application layer within development, QA, production
	€ 250 Total fee per month per application payable to in4it

All prices are excluding Belgian VAT (21%)

Includes:

Name	**Description**
Operating System Security updates	All Linux / Windows servers will receive operating system updates. Optionally, an agent can be installed which checks periodically for security vulnerabilities
Stand-by “best effort” SLA	The best effort Service Level Agreement doesn’t have any formal response times. You get a 24/7 telephone number to call to alert an engineer in case of any issues
Capacity Management	In4it will monitor the infrastructure and take action if any capacity issues arise.
Availability Management	In4it will monitor the production application for availability. If an unavailability event occurs, in4it will be notified and take action (using the “best effort” SLA)
Pro-active monitoring & alerts	Alerts will be set on some key metrics to make sure in4it is notified when inconsistencies would occur

Excludes:

Name	**Description**
Amazon AWS Services	Amazon AWS Services are charged directly to the customer