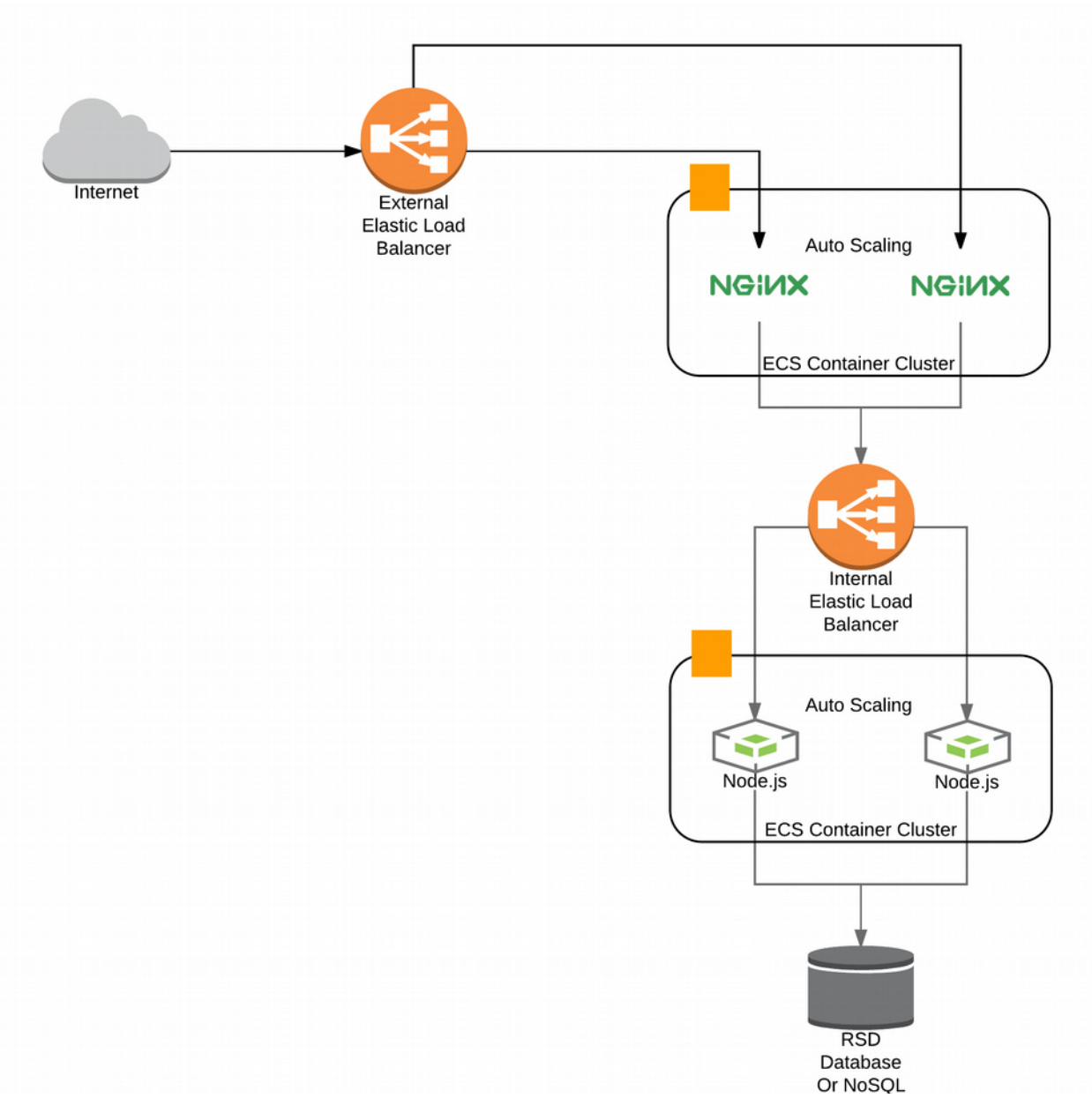
Deploying a web application

Nowadays it is hard to choose what services and architecture to use for continuos integration and deployment. There are plenty of good solutions; thus, usually the decision is based on costs, the level of security and reliability offered.

For this assignment, I have chosen AWS as it is the most widely used cloud services provider with an extensive variety of products to cover almost every scenario.

The most important aspects to bear in mind when deploying a web application are high availability, load balancing, 0-downtime application deployment, scalability and security. The following architecture diagram shows the integrations of several elements to cover these topics.



- High availability is guaranteed by the integration of Elastic Load Balancers and Amazon EC2 Container Service clusters. Those services automatically check the health of containers and take measures to increase resources or block an unhealthy node.
- The external an internal Elastic Load Balancers distribute the connection requests amongst the cluster of containers.
- Having different versions of the application and services by tagging the containers allows keeping old and new versions of the software running at the same time. This feature favours a gradual upgrade with no downtime during the deployment process.
- Autoscaling by setting a threshold for parameters or scheduling is a straightforward configuration that AWS provides the cluster products.
- Security has several levels with a critical one at the application layer, but having a firewall and SSL certificate on the web servers is essential. Amazon offers a Web Application Firewall, and NGINX, is one possible solution for a web server with SSL certificates.

Assignment Application Deployment

For the particular case of the application described in the assignment, the deployment could be done with this architecture by setting schedules and certain parameters to allow autoscaling:

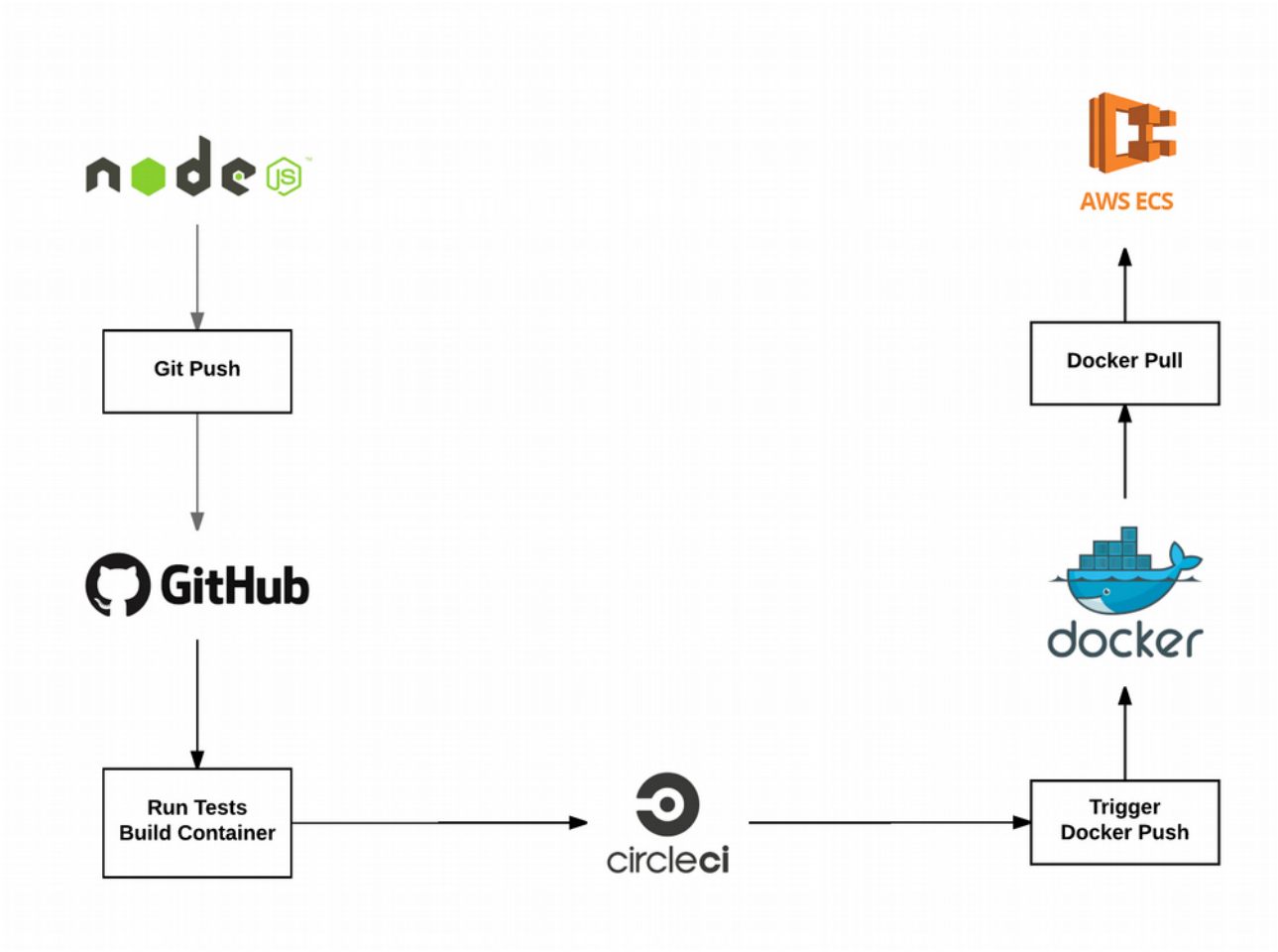
1. First, we would need to calculate how many connections per hour a single container running NGINX or an API, maybe coded with Node.js, can handle. For instance, if we say that one container can handle 500 connections per hour, we would need two instances to be reliable during normal hours. One would be enough, but high availability would be at risk if we rely on a single instance.
2. To cope with peak hours, we can create a schedule to increase the number of instances to 3 to have a capacity of up to 1500 connections per hour.
3. During Xmas a second schedule would need to increase the capacity to more than 40 instances.
4. Setting the Amazon CloudWatch to a certain threshold that triggers actions to autoscale the clusters of containers would ensure that the application keeps available under unpredictable circumstances.

Continuous Integration and Continuous Deployment

Another subject which takes advantage of these kinds of architectures using containers is continuous integration and deployment.

Automating the whole process of integrating code changes amongst a team of developers and deploying the code that has passed unit tests, helps to focus on solving user problems and delivering new features on a regular basis.

The diagram below shows a typical CI/CD cycle.



There are many more possible configurations; most of them share almost the same number of steps and features.

Extra comments

AWS can become and expensive solution if deployed applications have a high load of traffic. There are other alternatives to the AWS Elastic Load Balancer and EC2 Container Service:

- NGINX as a load balancer and reverse proxy.
- Iptables as a load balancer.
- Google Kubernets with pods of Docker containers.

I would be happy to discuss details about AWS or any other alternative.