Approach description & Scaling plan

Author: Guillermo Menéndez

Dec 25 2017

APPROACH DESCRIPTION

The developed solution is a cloud based solution with full automated HW and SW deployment and scaling.

It runs on EC2 instances using Docker containers and is deployed using Cloudformation templates.

What principles did you apply?

The following principles were applied in the solution design:

- Automation: environment deployments and updates are fully automated, as well as infrastructure scaling.
- Reproducibility: All resources are provisioned using infrastructure as code
- High availability: Infrastructure design and deployment policies try to achieve the minimum possible downtime
- Scalability: Applications scale automatically on high load conditions to guarantee availability and low response times
- Security: Cloud infrastructure and networking design provide resource isolation and access control

Explanation of the decisions you made and why

The following design decisions were taken:

- Use AWS cloud: in order to achieve a higher flexibility and automation level while keeping focus on the architecture design.
- Deploy each server as a standalone Docker container: this way VM content and installed libraries can be defined via code and easily deployed and replicated using standard tools.
- Use an ECS cluster to achieve easy scalability and redundancy, defining alarms to scale the infrastructure and services up and down.
- Deploy an application load balancer in front of the ECS cluster in order to route requests, leverage service-level autoscaling and provide a single entry point to the system.

What end state do you envision?

Current system environments reflect the intended end state, except for some fine tuning of deployment and scaling parameters in order to dimension the system for higher traffic loads. Also there is a critical issue that prevents system to scale correctly (see Scaling Plan section on this document).

System also could be improved following the recommendations for future work from this same section.

Why were certain tools selected?

Most tools were selected to achieve higher simplicity due to the assignment time constraints.

For this matter, most AWS services could have been replaced by Open Source tools, but lack of time for installing and configuring this tools lead to choose service-based options:

- Load balancing could have been achieved using an Open Source option, such as HAProxy, instead of Elastic Load Balancing.
- Docker orchestration could have been carried out using a Kubernetes cluster instead of EC2 Container Service.
- Infrastructure definition and deployment might have been performed using Terraform or Ansible instead of Cloudformation.

Why you configured the tools as you did?

Most of the tools used are services with small need for configuration in cases when standard parameters are fine to achieve the project goals.

Apache Tomcat and HTTPD servers were installed using their Alpine releases, which provide a thin layer of basic libs and options in order to leave the maximum possible resources to the deployed SW.

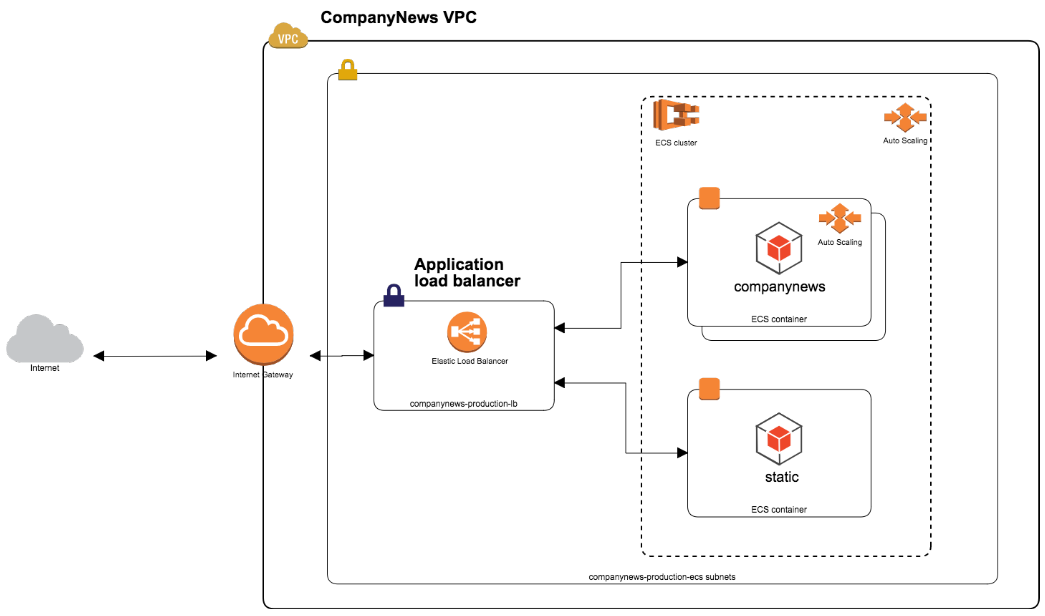
The critical configuration parameters for this project are the ones related to cluster and service dimensioning and deployment and scaling policies, which are further described in the Scaling Plan section of this document.

What is your recommendation for future work if time allows?

- Enforce HTTPS on the Load Balancer using a certificate, redirect HTTP requests to HTTPS.
- Configure custom domain URLs for each environment using Route53.
- Redirect Tomcat logs to Cloudwatch and configure logging filters and error alarms in order to automate process supervision.
- Remove the *static* Docker container and host static files on a CDN with geo caching enabled to achieve lower response times and higher scalability, while avoiding to maintain a dedicated server.
- Implement continuous deployment processes which leverage automatic deployment of the artifacts generated by developers into the training environment.
- Create an automatic pipeline which allows to deploy the SW running in the training environment to the production environment (for instance, a version control branch strategy using pull requests).
- Replace Prevayler persistence engine by any other persistence layer which allows horizontal scaling (see Scaling issues section at the end of this document).

SYSTEM ARCHITECTURE

Architecture design



Each of the system components are detailed next:

- **VPC:** all infrastructure components are instantiated inside the VPC for a higher isolation.
- **Internet Gateway:** allows VPC components to reach the internet and filters incoming connections.
- **Subnets:** three subnets in different availability zones containing the load balancer and the ECS cluster instances.
- **Application Load Balancer:** the load balancer routes incoming connections depending on the requested URL, thus allowing to have several docker containers listening behind the same IP and port.
- **ECS cluster:** the ECS cluster is a group of EC2 instances which act as hosts for docker containers. The cluster has associated autoscaling rules that create new instances when reserved memory is above a certain limit (80%).
- **CompanyNews container:** this container runs an Apache Tomcat application server together with the Company News web application. It has associated autoscaling rules that create new containers when CPU average usage is above certain limits (80%) for a period of at least 5 minutes. Autoscaling allows to have several containers serving the application users when traffic rises, and use less instances when traffic is lower.
- **Static files container:** this container runs an Apache HTTPD server serving the application static files. It's a single container with no autoscaling, as the static files should be cached to avoid problems with traffic peaks.

Infrastructure templates

The CloudFormation templates used to deploy the infrastructure are layered by components, so the code is clearer and easier to maintain. Each component template has its own parameters, and a main template performs the calls to every needed component with the appropriate values.

This is a brief overview of the designed CloudFormation templates:

- **companynews.json:** main template, glues together the project infrastructure and creates the needed IAM roles. Returns the Load Balancer endpoint, which is the system entry point from the internet.
- **vpc.json:** implements a VPC with an Internet Gateway.
- **az-subnets.json:** creates three VPC subnets in different availability zones.
- **load-balancer.json:** creates an internet facing Application Load Balancer with a default listener and target group.
- **ecs-cluster.json:** creates an ECS cluster with an autoscaling group and autoscaling policies, using the given instance type.
- **ecs-service.json:** creates a task definition and associated service from a docker image, includes service autoscaling and autoscaling policies.

SCALING PLAN

Limited release parameters

Current system parameters are defined for each environment having the limited release requisites in mind:

- **Docker cluster**
    - Min Docker cluster size: 1
    - Initial Docker cluster size: 1
    - Max Docker cluster size:
        - training: 1
        - production: 10
    - Scale up policies:
        - > 80% average CPU usage across cluster for 2 minutes
        - > 80% average memory reservation across cluster instances for 5 minutes
    - Scale down policies:
        - < 30% average memory reservation across cluster instances for 10 minutes
    - Update strategy:
        - Min instances in service: 0
        - Max batch size: 1

- **CompanyNews service**
    - Min task count: 1
    - Desired task count: 1
    - Max task count:
        - training: 1
        - production: 20
    - Scale up policies:
        - > 80% average CPU usage across service tasks for 2 minutes
    - Scale down policies:
        - < 20% average CPU usage across service tasks for 10 minutes
    - Deployment configuration:
        - Max percent: 100%
        - Min healthy percent: 0%

- **Static files service**
    - Min task count: 1
    - Desired task count: 1
    - Max task count: 1

Public release plan

Public release dimensioning heavily depends on the expected traffic volume. After the limited release is successfully conducted, measures should have been taking in order to refine the scaling and deployment policies and initial components sizing.

- **Docker cluster**
    - Docker cluster size:
        - Should be dimensioned taking into account the minimum and maximum traffic peaks and the required HW performance for the expected number of users.
        - Minimum size should be of at least 2 instances to ensure availability in case of HW failure.
    - Scale up policies:
        - More aggressive policies can be defined in order to perform HW upscaling before components are saturated.
        - It is recommended to define several scaling steps (for instance, scale 1 instance up if CPU usage is between 40% and 60%, scale 2 machines if usage is above 60%)
    - Update strategy:
        - Min instances in service: at least 2 instances in service at any moment
        - Max batch size: batch sizes can be increased if number of instances is high in order to speed upgrades up.

- **CompanyNews service**
    - Tasks count:
        - Should be dimensioned taking into account the minimum and maximum traffic peaks and the required memory and CPU requirements for the expected number of users.
        - Minimum size should be of at least 2 tasks to ensure availability in case of HW / SW failure.
    - Scale up policies:
        - More aggressive policies can be defined in order to perform service upscaling before SW reaches saturation point.
        - It is recommended to define several scaling steps (for instance, scale 1 instance up if CPU usage is between 40% and 60%, scale 2 machines if usage is above 60%).
        - It is recommended to add memory occupation upscaling policies, to avoid memory saturation.
    - Deployment configuration:
        - Max percent: Should be between 150% and 200% percent to avoid downtimes or performance loss during SW updates.
        - Min healthy percent: 100%
        - It is recommended to implement a blue-green deployment strategy, in order to reduce downtime risk and allow rollbacks in case of SW failure.

- **Static files service**
  - It is recommended to implement a caching system or use a CDN + geocaching in order to avoid serving the same static files from a server for every request.

Scaling issues

A scaling issue has been detected in the CompanyNews application. This issue is critical and makes currently impossible to horizontally scale the application, thus only being able to scale by means of using higher performance machines.

The Prevayler persistence layer is single-host based, thus rendering impossible to have several instances of the CompanyNews application running in different hosts. Prevayler stores all objects in memory and logs transactions to disk, all data remaining local to the server host.

In order to solve this issue, two approaches can be taken:
- Enable a synchronization mechanism amongst instances for Prevayler persistence engine.
- Use a different storage engine, such as MySQL, DynamoDB or any other SQL or NoSQL solution.