Container-based Serverless Web Hosting with Amazon Fargate

Andrew Kaczorek, Chris Chalfant

Leaf Software Solutions

Who Are We and What Do We Do?

Leaf Software Solutions:

- Located in Carmel
- In business for over 30yr

Project categories:

- Custom enterprise software development
- Open source development for MVPs, startups
- Cloud/Devops consulting and architecture
- Microsoft GP/Dynamics accounting/CRM solutions

Our “Pre Legacy” Approach to Hosting

- Generally on-prem environments
- Typically managed by IT teams
- Occasionally Leaf provides some support
- Upgrades and patching performed ad-hoc
- Automate where we can with CI/CD
  - Teamcity
  - Octopus Deploy

Our “Legacy” Approach to Hosting

- Heavily automated environments running in IaaS (mostly EC2)
- Technologies:
  - CloudFormation, Terraform for cloud resources
  - Chef or other cfg management
  - Hashicorp Packer for base images
  - CI/CD in TeamCity or Docker-based (Bitbucket Pipelines, Gitlab)
- Ongoing support is usually replacement of resources with newer versions
- Still requires a certain amount of traditional sysadmin

No Servers: A Good Fit for Leaf

- By “no servers”, we mean no remote (RDP/SSH) access
  - We don’t want to manage instances/virtual machines
  - We aren’t staffed for traditional OS-level support. Some of our clients aren’t either.
- Use SaaS for everything else we need:
  - Build/deploy environments
  - Performance and availability monitoring
  - Log aggregation, event alerting
  - Misc services (email, auth, business analytics, etc)

First, Let's Talk Serverless (Function-as-a-service)

- Leaf has production experience with a true Serverless app
- This approach has some unique benefits:
  - Instantaneous and vast scalability if configured correctly
  - Any kind of OS configuration is not even relevant to these systems
  - Truly pay for what you use, which is generally very little
- This approach has some unique challenges:
  - Generally more complicated dev, typical challenges of "emerging" techniques
  - Can be hard to troubleshoot what is happening *right now*
  - Having huge scalability can cause problems hitting constrained resources:
    - Relational databases (cpu, db connection #, etc)
    - Network resources in your VPC
    - Throttling of APIs and 3rd party services

Containers, a quick reminder

- Package and run an application as one (or very few) services in a container
- A container shares OS resources with other containers
- Kernel-level support for access restriction, CPU and memory limits
- Containers run on one or more hosts and are managed by a scheduling system.
- Scheduling systems decide where, when, and how containers are distributed on hosts.

Container-based App Hosting

- We use AWS ECS Fargate, but this applies to any docker scheduler on any platform
- Build and deploy a container image with code
  - In a hosted build env, build a container within a container (yo dawg)
  - The container is the build artifact
- Inject config at deployment time (12-factor app) -- no config built in.
- Same container can be used in dev, test, prd

Container Build Process

- Note: Our reference environment for this presentation is Dotnet Core:
  - The resulting containers run Linux like other envs: Django, Rails, etc
- Pipelines triggers build in our customized Microsoft Dotnet Core container
- Our dotnetcorebuild container builds the app:
  - https://github.com/LeafSoftware/dotnetcorebuild
  - https://hub.docker.com/r/leafsoftware/dotnetcorebuild/
  - Our intent for this general purpose container is to build/deploy .NET core apps
- This ubuntu-based application container also:
  - Installs app, runs docker build, docker push
  - Registers ECS database migration task and runs it
  - Registers new ECS task for the app and updates the service

ECS Fargate Web Hosting Features

- Horizontally scalable: Fargate schedules containers for us, over multiple AZs and keeps them running
- Load-balanced: Application Load Balancers with tight Sec Groups
- Fargate registers/de-registers containers with load balancer
- Zero downtime deployments: configure minimum number of running containers and how many to replace at a time. System drains requests before stopping containers.

ECS Anatomy

One or more containers run as a Task (configured by a Task Definition)

Tasks run on a Cluster (a set of hosts running the ECS agent)

Tasks can be one-off or on-going.

One or more on-going Tasks are configured as a Service

With Fargate, AWS manages the hosts, you don’t interact with them at all.

Detailed Workflow of ECS / Fargate

1. Create a Cluster
2. Register a Task
    a. CPU and RAM
    b. Container Image(s)
    c. Port Mapping, Volumes
    d. IAM Role
3. Create a Service
    a. Task, number of tasks
    b. Load Balancer
    c. VPC, subnets, security groups

Deploying a New Version in ECS/Fargate

1. Build and Push the new container
2. Run the database migration first (as an ECS/Fargate task itself)
    a. Deployment is halted if the migration fails
    b. Avoids running DB in public subnet (AWS-managed laptop-to-VPC VPN would be nice!)
3. Register new app Task Definition with new Container Image
4. Update Service to use new Task Definition

ECS drains requests and replaces running containers according to your Deployment Configuration rules.

Secrets Management

- Initially we used Bitbucket Pipelines env variables
- We have recently started using Chamber by Segment
- This uses AWS Systems Manager Parameter Store
- Our containers have fine-grained IAM access to the secrets
- Secrets are encrypted using KMS which is awesome
- AWS Secrets Manager just released but it’s super pricey

Logging & Monitoring in ECS/Fargate

- Easy to shunt Container STDOUT to CloudWatch Logs, then on to Loggly--or direct to Loggly
- Your typical app-monitoring solutions can run inside of containers just fine (New Relic, Datadog, raw Cloudwatch, etc)
- There are some forensics of a stopped container that are ephemeral in AWS much like a terminated EC2 instance

Why Is This Powerful?

- No OS to manage for build system, app servers, or databases. In the shared responsibility environment, AWS is responsible for managing the host OS. This means:
  - No AMIs to manage
  - No traditional anti-virus required (many regulated .NET Windows require this)
  - No CFG mgmt or Octopus Deploy required: rolling deploys are automatic and done for you
  - No Reserved Instance capacity planning

What Are Some Gotchas?

- It can be difficult to keep straight all of the:
  - Layout inside of the various containers
  - App paths, working directories
  - Entry points, default docker run commands
  - Containers running in Bitbucket vs containers running in VPC
- Pruning of old containers, definitions, log groups, etc
- Troubleshooting can be tricky
- At the end of the day, the container still looks like an OS
  - Even though it is just (typically) one running process, it is still sits inside of a traditional OS container layout with config, packages, tmp state, etc

Fargate vs True Serverless?

- Fargate
  - Use existing code, frameworks
  - Is expensive (3x cost of running your own instances)
  - Containers a bit slower to spin up
- Serverless (FaaS)
  - Good for event-based systems. Still complicated for web/API
  - Programming paradigm can have higher learning curve

A happy medium?

- The ECS/Fargate seems a good compromise between doing things the “old way” and going all-in on Serverless.
  - Easier to hand off development and/or operations to less experienced staff
  - For Dotnet specifically, there is a smooth transition from building traditional .NET apps with Visual Studio and going to ECS
  - Portability is an easy story with ECS. Want to run on your laptop? Want to move it on-prem? Want to take it to another cloud provider? All very simple here thanks to Docker.

The Future of App Hosting @ Leaf

- ECS + Fargate is going to be our prefered architecture for the near future
- Next up: Cloud-native serverless databases (Dynamo, Aurora, etc)
- Serverless is a fantastic fit for event-based needs such as processing items that appear in a bucket, consuming a queue of work, etc
- It is likely that capital-S-Serverless and container-based approaches will converge a bit:
    - For example, Django/Flask apps in Zappa can run on a traditional app server or in Lambda with some caveats
    - Even though a docker container is much lighter than a VM, we are still carrying around a lot of bits in the container for the OS that encapsulates the process(es)

Parts List for our Reference Environment

Source Code Control: Bitbucket

Container Registry: AWS EC2 Container Registry (private), Docker Hub (public)

Build System: Bitbucket Pipelines

Networking: Virtual Private Cloud, Application Load Balancer, Route 53

Storage: Relational Database System

Container Schedler: AWS Elastic Container Service

Fargate POC Price Analysis

Development Environment:

- Load balancer: $0.0225 per Application Load Balancer-hour
- **App server: $0.152 per hour for two 1 core / 2GB RAM containers**
- RDS instance: $0.29 per hour for multi-AZ t2.large Postgres RDS instance
- Total: $0.4645 per hour or $334/mo

Hypothetical Production Environment:

- Load balancer: $0.0225 per Application Load Balancer-hour
- App server: $0.304 per hour for two 2 core / 4GB containers
- RDS instance: $0.36 per hour for multi-AZ m4.large Postgres RDS instance
- Total: $0.6865 per hour or $494/mo

Compare to:

- **Pair of t2.small (1 CORE / 2GB RAM) running 24/7: $0.046 per hour**
    - Same capacity of Fargate costs about 3.3x more in this case
- Single Multi-AZ SqlServer Standard m4.large RDS: $1.955 per hour or $1408/mo
    - Many .NET projects use SqlServer, so this number dwarfs the others

Thanks For Listening

- IndyDevops Meetup is once a month on the third monday @ 7pm:
  - https://www.meetup.com/IndyDevOps
  - Dinner is provided
  - Always looking for speakers
- Devops Days Indianapolis is July 23-24:
  - https://www.devopsdays.org/events/2018-indianapolis/welcome/
  - Looking for attendees and sponsors
  - We have $20 off coupons!
- Feel free to ask us questions here or reach out:
  - akaczorek@leafsoftwaresolutions.com
  - cchalfant@leafsoftwaresolutions.com