CON 310

Moving to Containers:
Building Using Docker and Amazon ECS

Uttara Sridhar, Software Development Engineer, AWS
Thilina Gunasinghe, Senior Director, McDonald's Corporation
Manjeeva Silva, Technical Architect, McDonald's Corporation

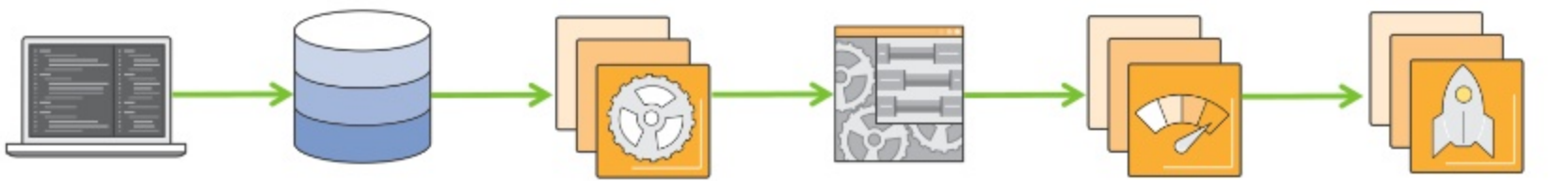
AWS re:Invent

aws

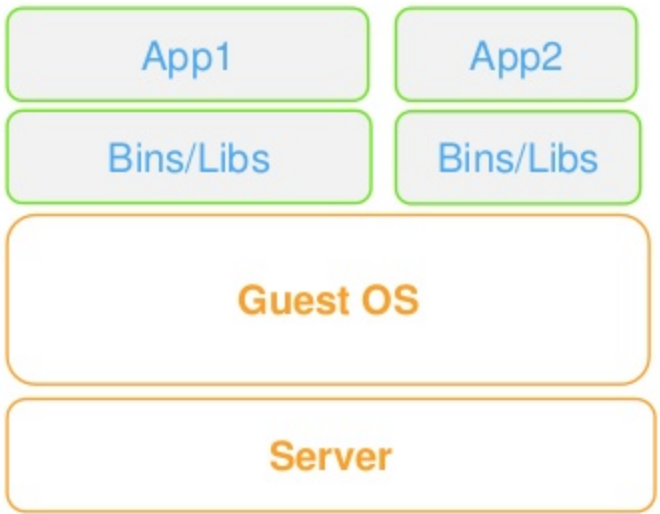
What to Expect from the Session

- Review software development lifecycle
- Why move to containers
- Setup image repository with Amazon ECR
- Deploy to Amazon ECS
- Log, monitor and scale containers
- Automate using CI/CD
- Deep Dive McDonald's home delivery platform

Software Development Lifecycle



Why Move to Containers?



* Portable
* Flexible
* Fast
* Efficient

Containerize Code

Dockerfile

```
FROM ubuntu:12.04
# Install dependencies: apache
RUN apt-get update -y
RUN apt-get install -y apache2
# write hello world message
RUN echo "Hello World!" > /var/www/index.html
# Configure apache
RUN a2enmod rewrite
RUN chown -R www-data:www-data /var/www
ENV APACHE_RUN_USER www-data
ENV APACHE_RUN_GROUP www-data
ENV APACHE_LOG_DIR /var/log/apache2
EXPOSE 80
CMD ["/usr/sbin/apache2", "-D",  "FOREGROUND"]
```

Dockerfile Best Practices

- Reduce build image sizes and number of layers
- Only bring what you need at runtime
- Always tag each build with any of these schemes:
    - Semantic version (i.e. “1.3.2-9”)
    - Git SHA (i.e. “aebcbca”)
    - Build Number (i.e., “127”)
    - Build Id (i.e. “511d5e51-b415-4cb2-b229-b3c8a46b7a2f”)
- For multi-region, push to all regions, pull from same region

Docker Build

Docker Images

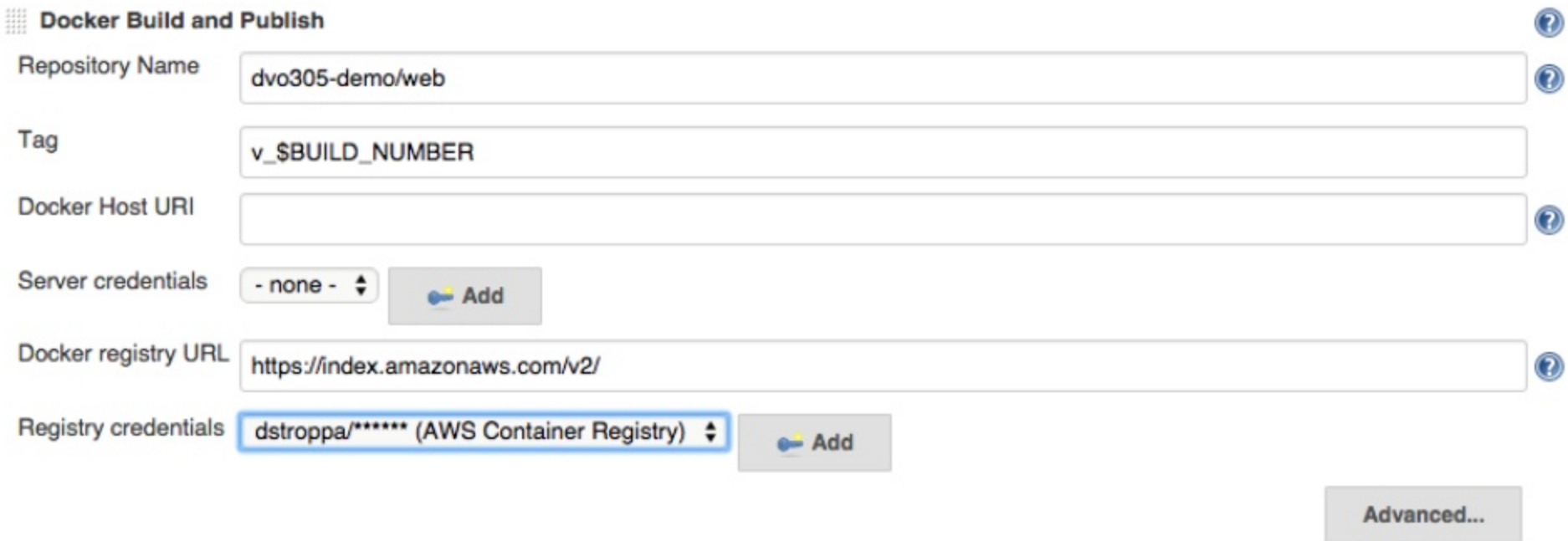
Packaged Application Code

Reproducible

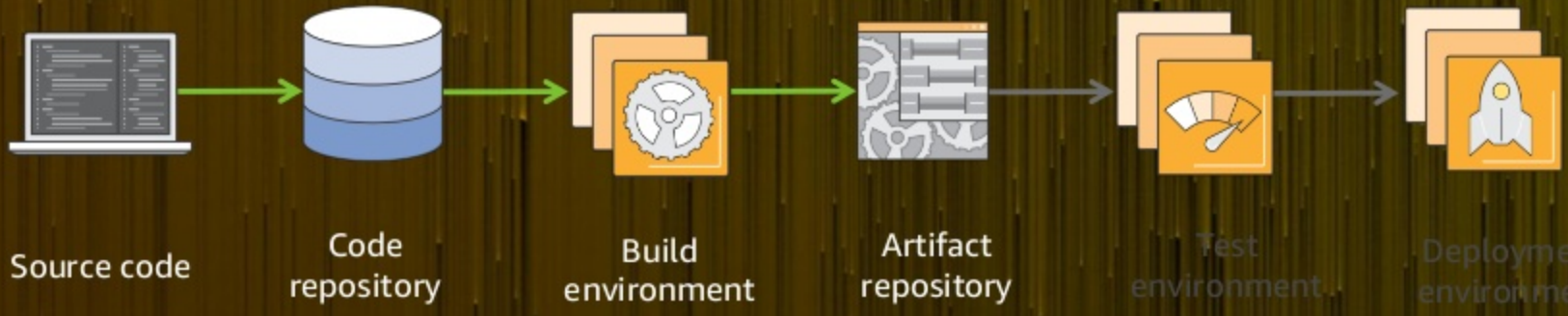
Immutable

Portable

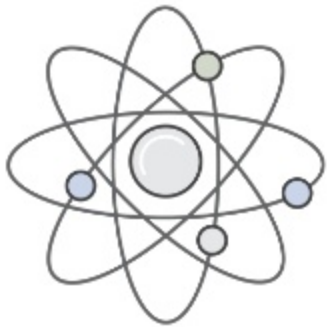
Docker Build (Using Jenkins)



Amazon ECR: Container Image Registry



Amazon Elastic Container Registry



Fully Managed

* Tight Integration with Amazon ECS
* Integration with Docker Toolset
* Management Console and AWS CLI

Highly Available

* Amazon S3 backed
* Regional endpoints

Secure

* IAM Resource-based Policies
* AWS CloudTrail Audit Logs
* Images encrypted at transit and at rest

Amazon ECR Setup

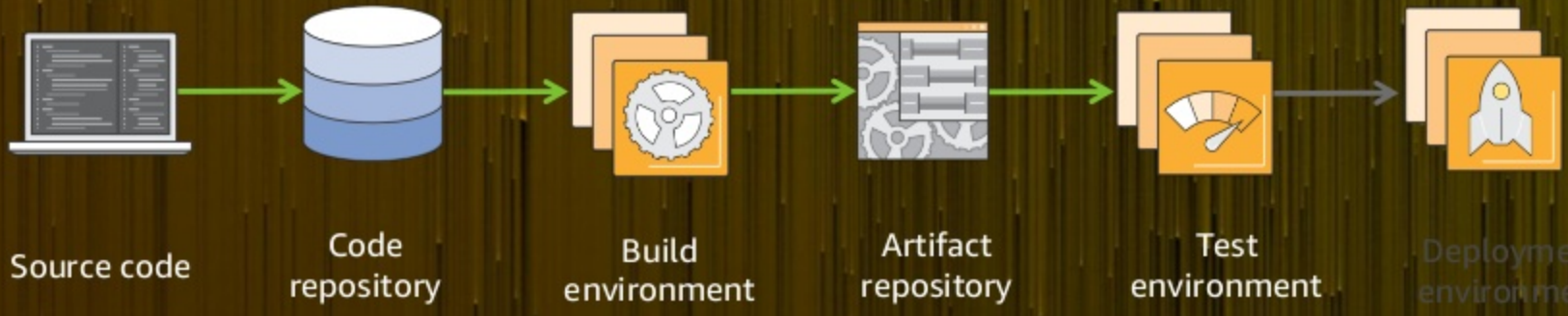
```
# Create a repository called hello-world
> aws ecr create-repository --repository-name hello-world

# Build or tag an image
> docker build -t hello-world .
> docker tag hello-world aws_account_id.dkr.ecr.us-east-1.amazonaws.com/hello-world

# Authenticate Docker to your Amazon ECR registry
> aws ecr get-login
> docker login -u AWS -p <password> -e none aws_account_id.dkr.ecr.us-east-
1.amazonaws.com
# You can skip the `docker login` step if you have amazon-ecr-credential-helper setup.

# Push an image to your repository
> docker push aws_account_id.dkr.ecr.us-east-1.amazonaws.com/hello-world
```

Test


Source code
Code repository
Build environment
Artifact repository
Test environment
Deployment environment

Running Test

- Inside the container:

Usual Docker commands available within your test environment

Run the container with the commands necessary to execute your tests, e.g.:

```
docker run web bundle exec rake test
```

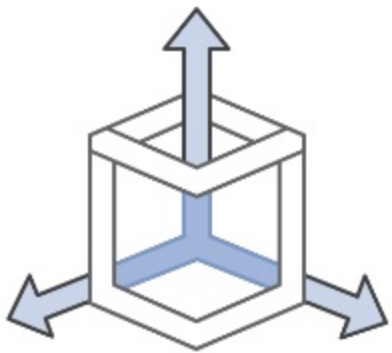
- Against the container:

Start a container running in detached mode with an exposed port serving your app

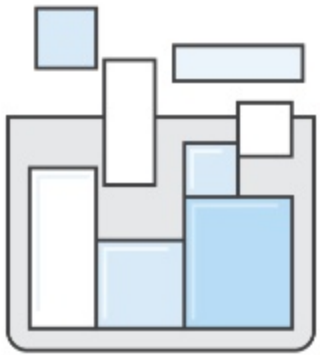
Run browser tests or other black box tests against the container, e.g. headless browser tests

Deploy to Amazon ECS

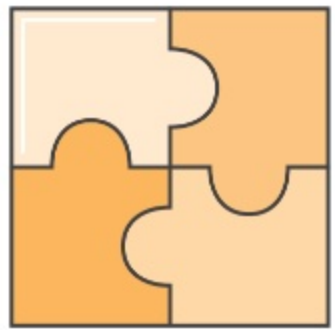
Amazon Elastic Container Service



Easily manage clusters of any size



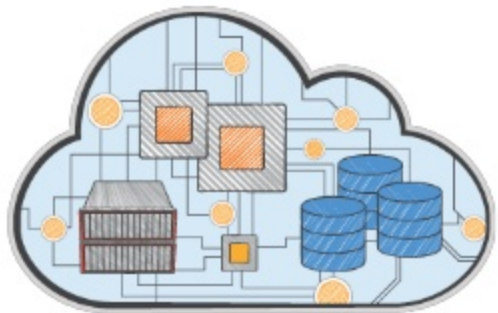
Flexible Container Placement



Integrated with AWS services

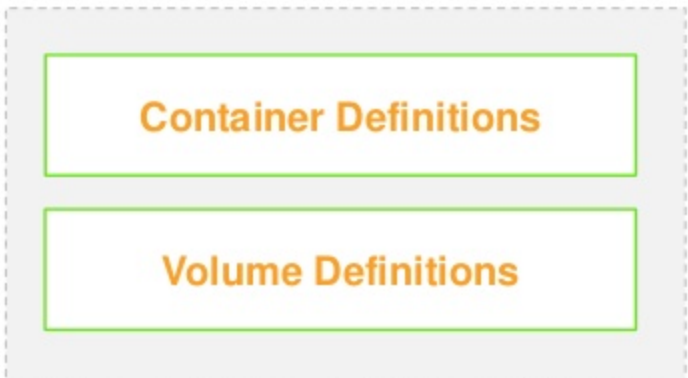
Extensible

Amazon ECS Core Concepts



- Cluster
- Container Instances
- Task Definition
- Task
- Service

Amazon ECS Task Definition




```
{
  "family":"hello-world",
  "containerDefinitions":[
    {
      "name":"hello-world",
      "image":"aws_account_id.dkr.ecr.us-east-
1.amazonaws.com/hello-world",
      "cpu":10,
      "memory":500,
      "portMappings":[
        {
          "containerPort":80,
          "hostPort":80
        }
      ],
      "entryPoint":[
        "/usr/sbin/apache2",
        "-D",
        "FOREGROUND"
      ],
      "essential":true
    }
  ],
  "volumes": []
}
```

Amazon ECS Service

```
{
    "serviceName": "ecs-hello-world-elb",
    "taskDefinition": "hello-world",
    "loadBalancers": [
        {
            "loadBalancerName": "EC2Contai-
EcsElast-A0B1C2D3E4",
            "containerName": "hello-world",
            "containerPort": 80
        }
    ],
    "desiredCount": 10,
    "role": "ecsServiceRole"
}
```

* Good for long-running applications
* Load Balance traffic across containers
* Automatically recover unhealthy containers
* Discover services

Amazon ECS Service

- Deployment
- Logging
- Monitoring
- Scaling

Rolling Deployments

```
{
    "serviceName": "ecs-hello-world-elb",
    "taskDefinition": "hello-world",
    "deploymentConfiguration": {
        "maximumPercent": 100,
        "minimumHealthyPercent": 50
    },
    "loadBalancers": [
        {
            "loadBalancerName": "EC2Contai-EcsElast-A0B1C2D3E4",
            "containerName": "hello-world",
            "containerPort": 80
        }
    ],
    "desiredCount": 10,
    "role": "ecsServiceRole"
}
```

Rolling Deployments

minimumHealthyPercent = 50%, *maximumPercent* =100%

Rolling Deployments

minimumHealthyPercent = 100%, *maximumPercent* = 200%

Amazon ECS Service

- Deployment
- Logging
- Monitoring
- Scaling

Logging with Amazon CloudWatch Logs

- Supported Docker Logging Drivers
  - json-file, syslog, journald, gelf, fluentd, awslogs
  - stdout/stderr outputs are automatically sent by the driver

- awslogs sends logs to Amazon CloudWatch Logs
  - Log group for a specific service
  - Log stream for each container

- Amazon CloudWatch Logs => Other services
  - Search, Filter, Export to Amazon S3, Send to Amazon Kinesis, AWS Lambda, Amazon Elasticsearch Service

Configuring Logging in Task Definition

```
{
  "family":"hello-world",
  "containerDefinitions":[
    {
      "logConfiguration":{
          "logDriver":"awslogs",
          "options":{
            "awslogs-group":"awslogs-test",
            "awslogs-region":"us-east-1",
            "awslogs-stream-prefix":"hello-world"
        }
      },
      "name":"hello-world",
      "image":"aws_account_id.dkr.ecr.us-east-1.amazonaws.com/hello-world",
      "cpu":10,
      "memory":500,
...
```

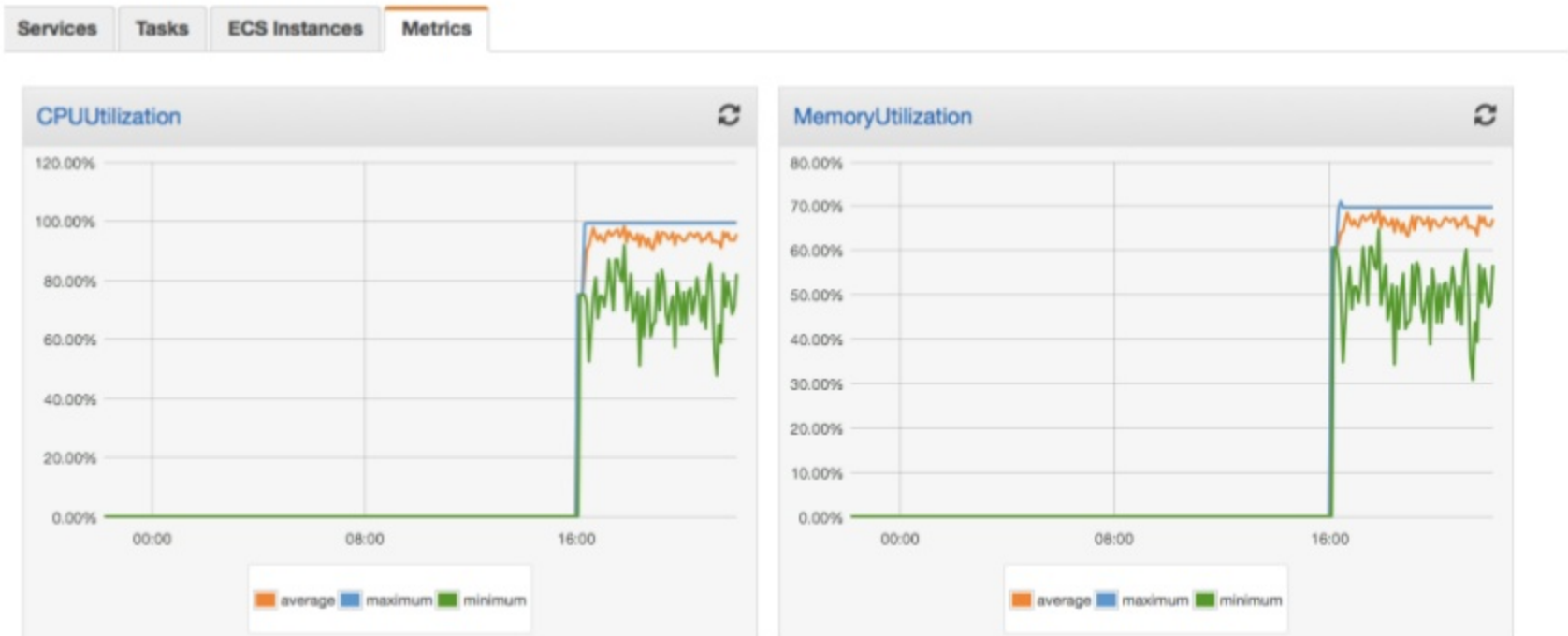
Amazon ECS Service

- Deployment
- Logging
- Monitoring
- Scaling

Monitoring with Amazon CloudWatch

- Metric data sent to CloudWatch in 1-minute periods and recorded for a period of two weeks
- Available metrics:
    - CPUReservation
    - MemoryReservation
    - CPUUtilization
    - MemoryUtilization

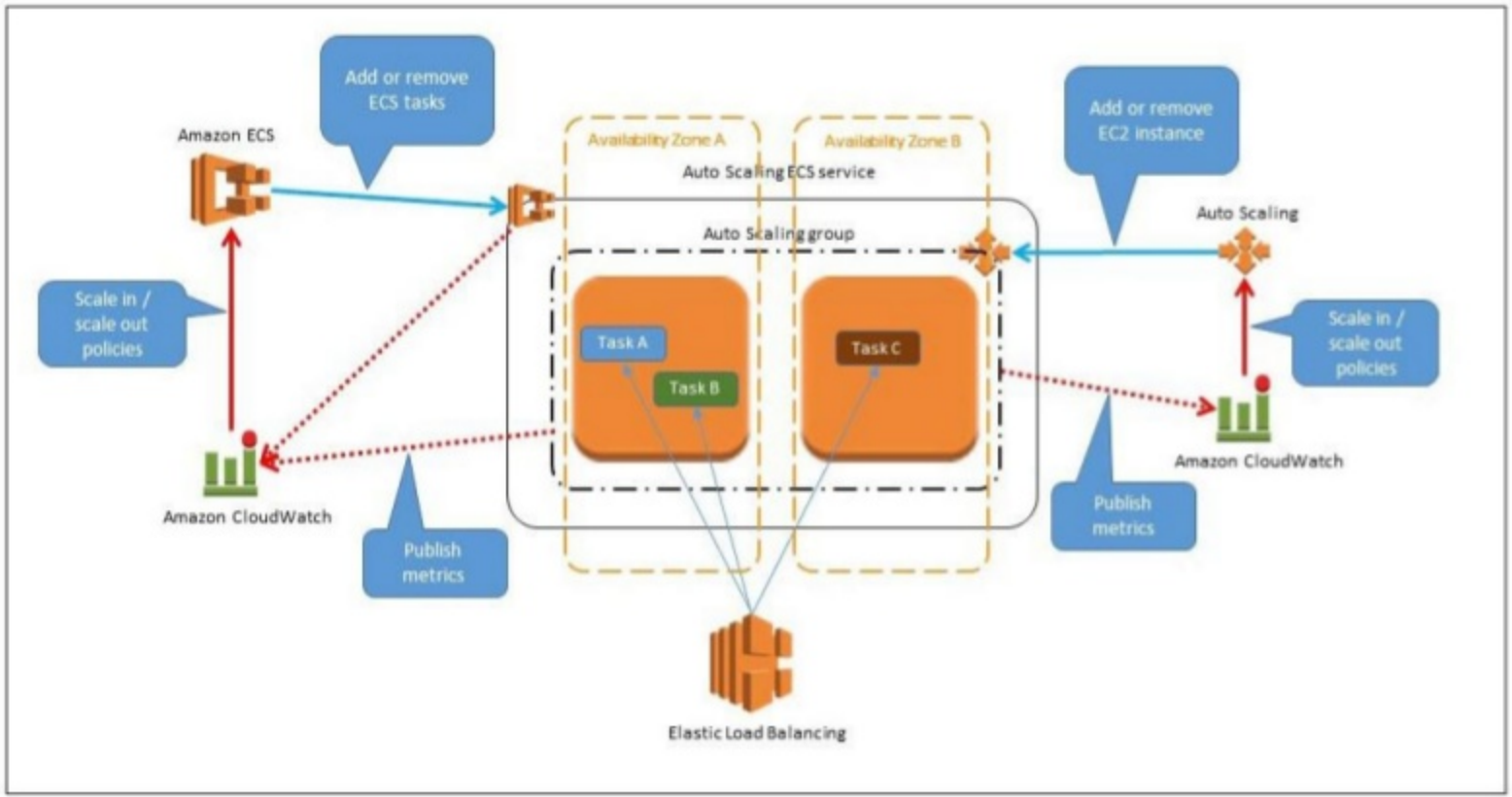
Monitoring with Amazon CloudWatch



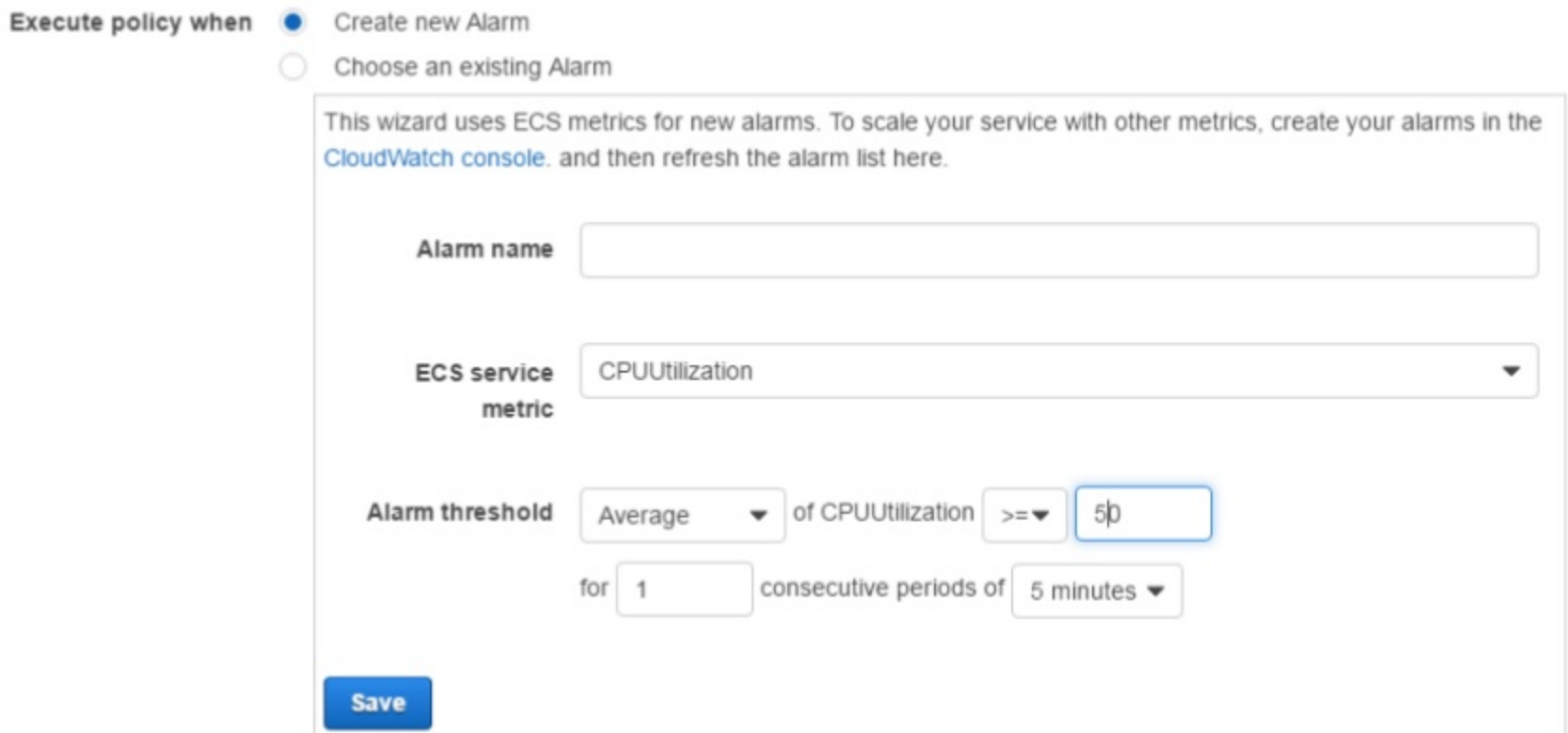
Amazon ECS Service

- Deployment
- Logging
- Monitoring
- Scaling

Auto Scaling Your Amazon ECS Resources



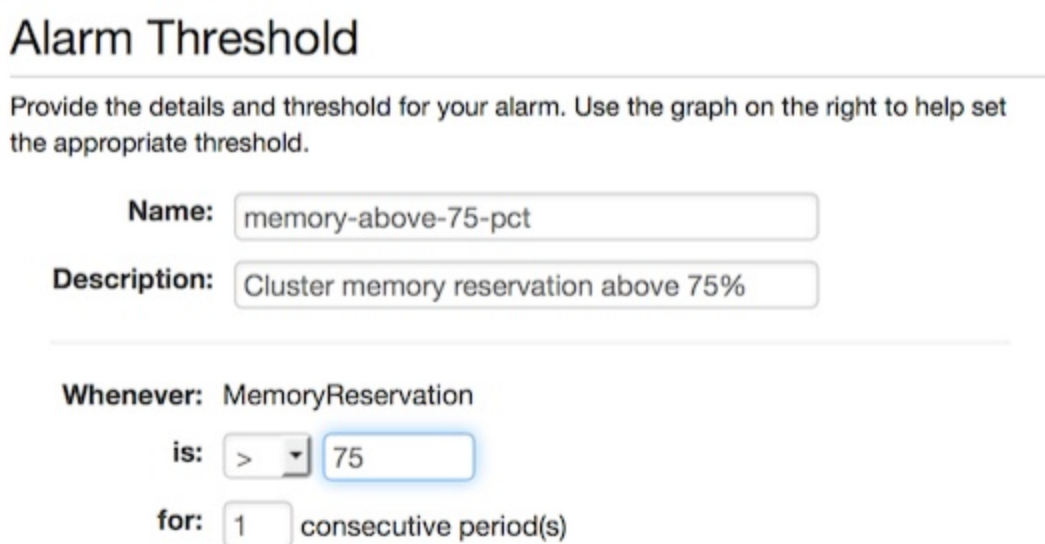
Auto Scaling Your Amazon ECS Services



Auto Scaling Your Amazon ECS Cluster

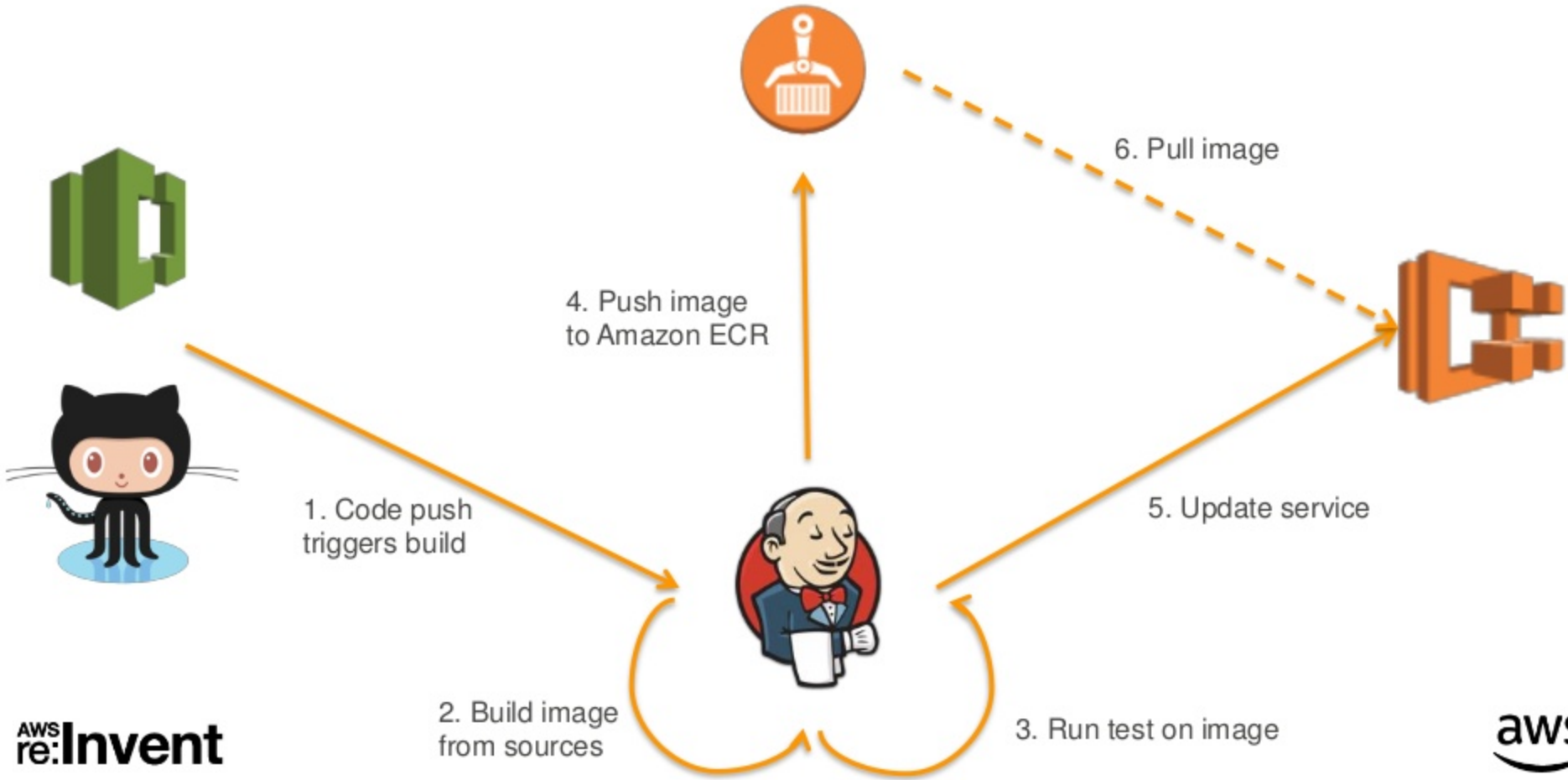
Create Amazon CloudWatch alarm on a metric, e.g. MemoryReservation

Configure scaling policies to increase and decrease the size of your cluster

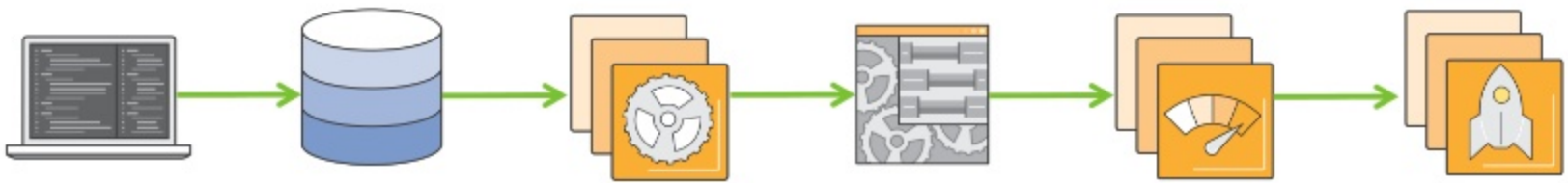


Automate Using CI/CD

Continuous Delivery to Amazon ECS with Jenkins



Summary



Source code	Code repository	Build environment	Artifact repository	Test environment	Deployment environment
Dockerfiles	-AWS CodeCommit -Github	-AWS CodeBuild -Jenkins	-Amazon ECR -Dockerhub	-Amazon ECS -Jenkins	-Amazon ECS

CI/CD Partners

- -Jenkins
- -Shippable
- -Codeship
- -wercker
- -Solano Labs

CON 310

AWS re:INVENT

McDonald's – Moving to Containers

Thilina Gunasinghe and Manjeeva Silva

AWS re:Invent

aws

Contents

- About **McDonald's** and **Digital Acceleration**
- McDonald's **Home Delivery Platform**: Business Problem and Architecture
- Under the Covers: How we used ECS for **Scale, Speed, Security, DevOps and Monitoring**
- Final Thoughts and **Takeaways**

About McDonald's and Digital Acceleration

WORLD'S LARGEST RESTAURANT COMPANY

37K
RESTAURANTS

1.9M
PEOPLE
Working for McDonald's & Franchisees

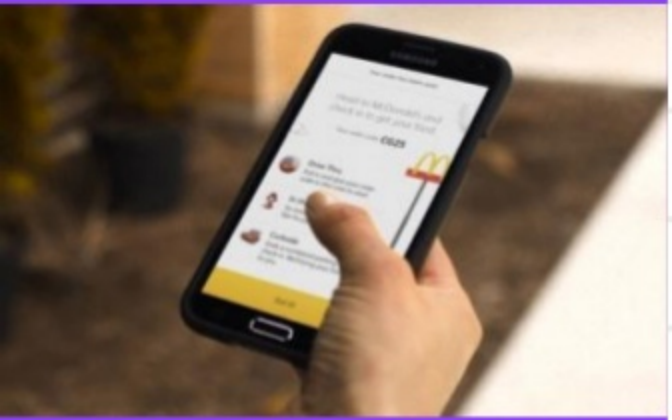
120
COUNTRIES

64M+
CUSTOMERS
Served Every Day

VELOCITY ACCELERATORS

McDonald's uses scale as a competitive advantage to surpass the rising expectations of our customers across three growth initiatives.

digital



Seamless, personalized engagement with our customers when they are at home, on-the-go, or in a restaurants.

delivery

Bringing McDonald's directly to customers.

experience of the future

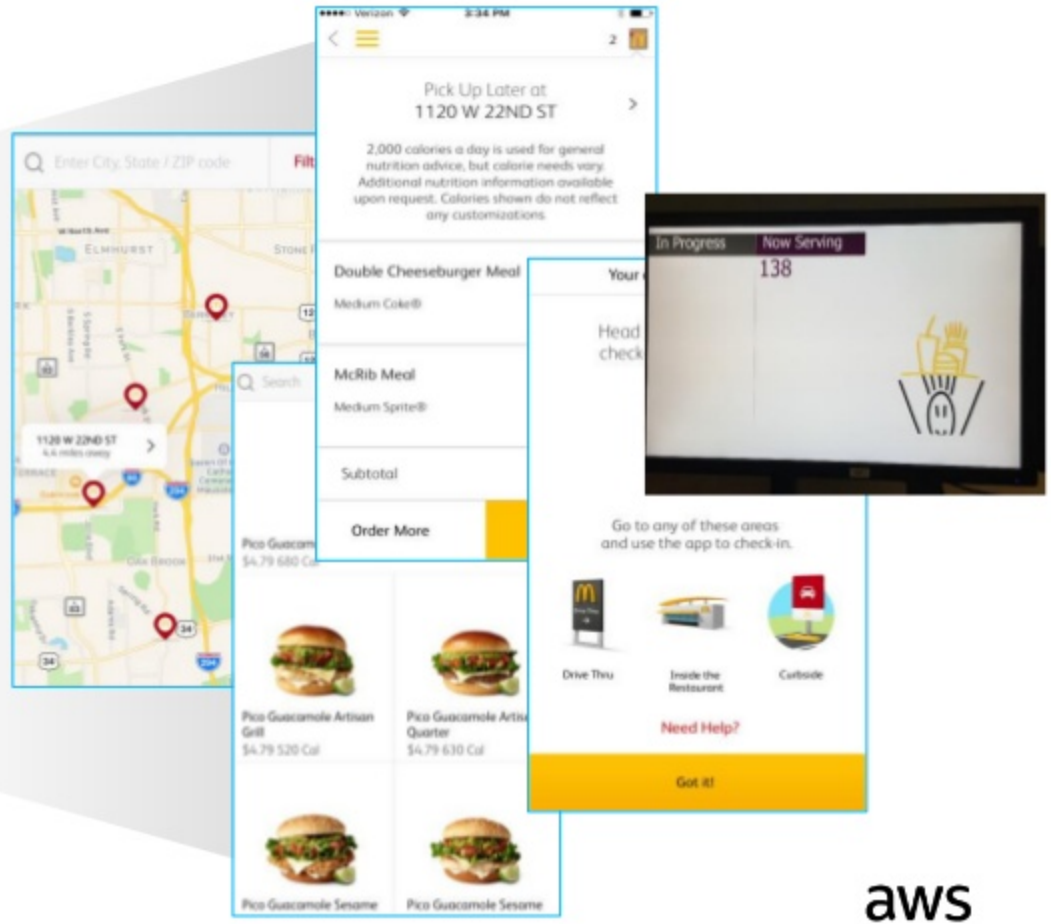
Elevating the McDonald's restaurant experience.

McDonald's Home Delivery Platform: Business Problem and Architecture

Introducing Home Delivery

digital & delivery

Bringing McDonald's to you



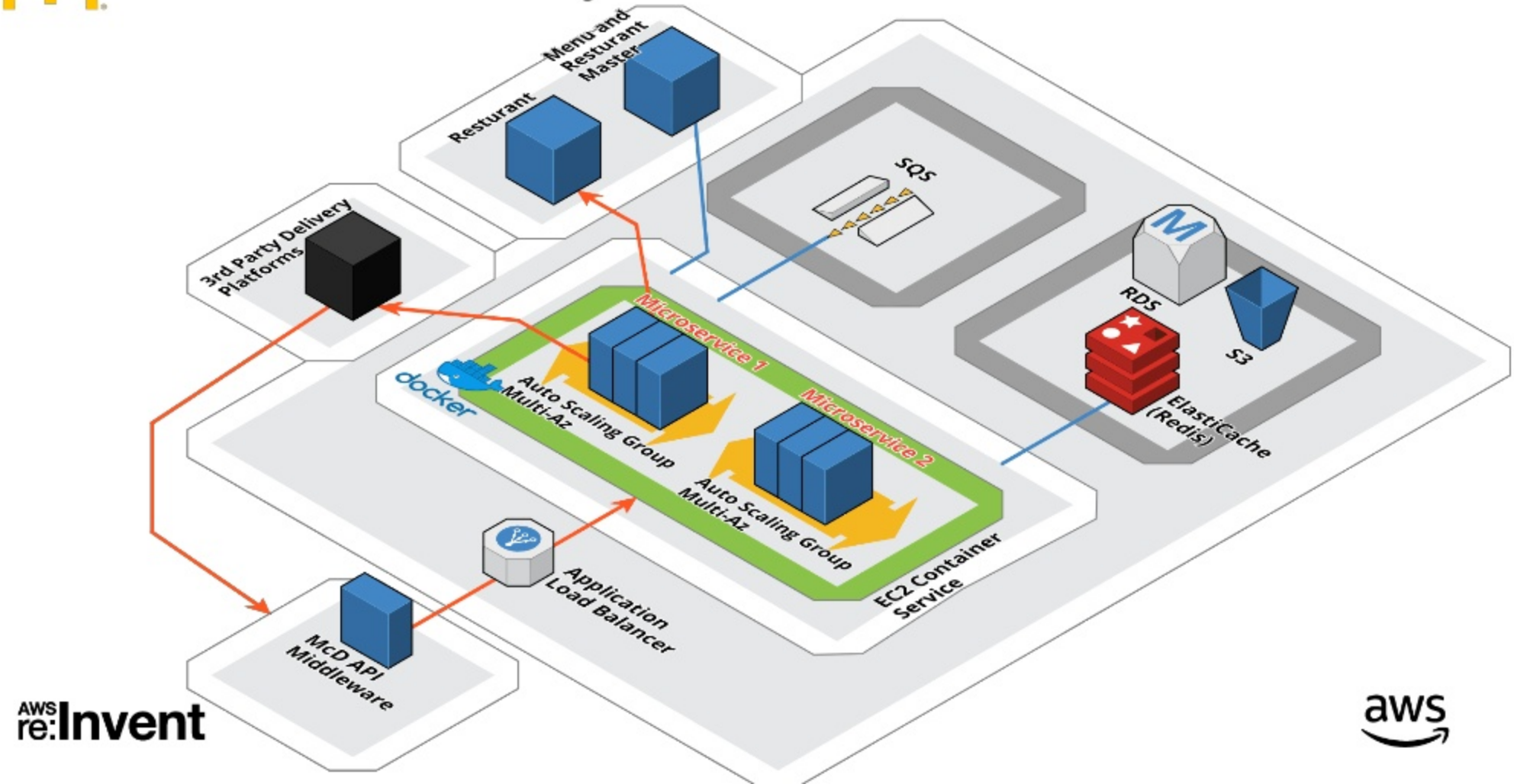
AWS re:Invent

aws

Critical Business Requirements

- **Speed to market**, quick turn around for features and functionality from concept to production
- **Scalability and reliability** targets of, 250K-500K orders per hour
- **Multi country support** and integration with multiple 3rd party food delivery partners
- **Cost sensitivity**, cost model based on low average check amounts

Home Delivery Architecture

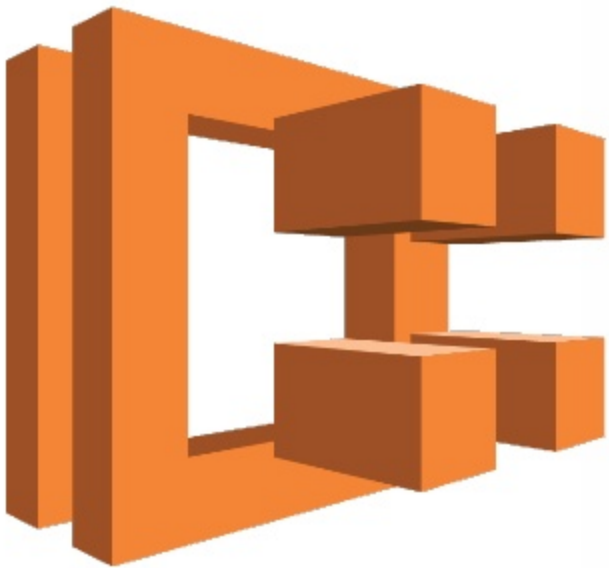


Key Architecture Principles

- **Microservices**, with clean APIs, service models, isolation, independent data models and deployability.
- **Containers and orchestration,** for handling massive scale, reliability and speed to market requirements
- **PaaS,** based architecture model by leveraging AWS platform components such as ECS, SQS, RDS and Elasticache
- **Synchronous and event based**, programming models based on requirements

Under the Covers: Using Amazon ECS for Scale, Speed, Security, DevOps and Monitoring

Why Amazon ECS?



- Speed to market
- Scalability and reliability
- Security
- DevOps – CI / CD
- Monitoring

Speed to Market

- **4 months** from concept to production with 2 week dev iterations
- **Polyglot tech stack** ported to containers
    - Existing .net code was refactored and complied with .net core
    - Java was used where .net core is not supported
- **Simplified Amazon ECS deployment model** with easy integration to AWS services
- **Less time** was spend on **application tuning/testing** to achieve the scale and reliability requirements
- **DevOps** – Faster feedback loops on release iterations

Scalability and Reliability

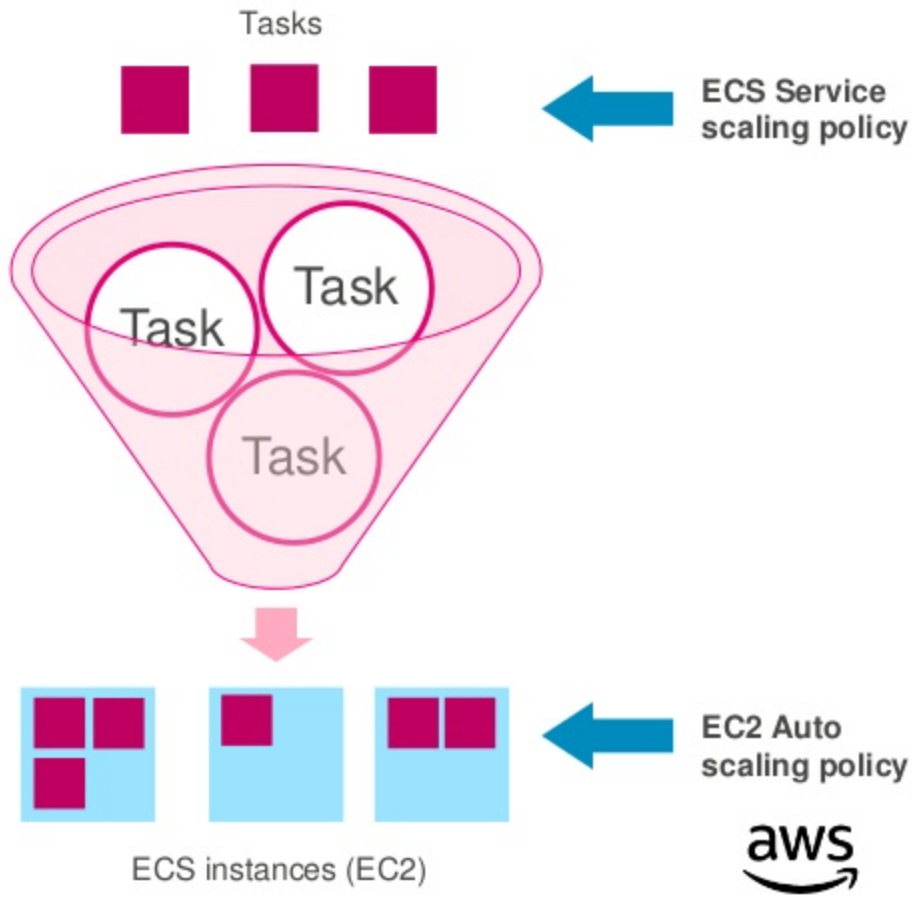
- **Scale Targets:** Achieved the scale targets of 250k-500k orders per hour with below ~100ms response times
- **Autoscaling:** Used Amazon ECS “out-of-box” resource based autoscaling
- **Task Placement:** Task placement strategies and constrains were used to fine-tune and achieve container isolation with country / 3rd party scaling requirements
- **Scalability and Reliability Requirements (By Service):**
    1. {Service 1} Contains synchronous APIs with intense scalability and reliability requirements based on traffic burst patterns
    2. {Service 2} Requires more batch mode processing. Work load optimization is a critical requirement
    3. Some instances of {Service 3} will need isolation from others

Scalability and Reliability

```
{Service 1} Task Definition
"placementStrategy": [
    {
        "field": "attribute:ecs.availability-zone",
        "type": "spread"
    }

{Service 2} Task Definition
"placementStrategy": [
    {
        "field": "memory",
        "type": "binpack"
    }
]

{Service 3} Task Definition
"placementConstraints": [
    {
        "expression": "task:group == US",
        "type": "memberOf"
    }
]
```

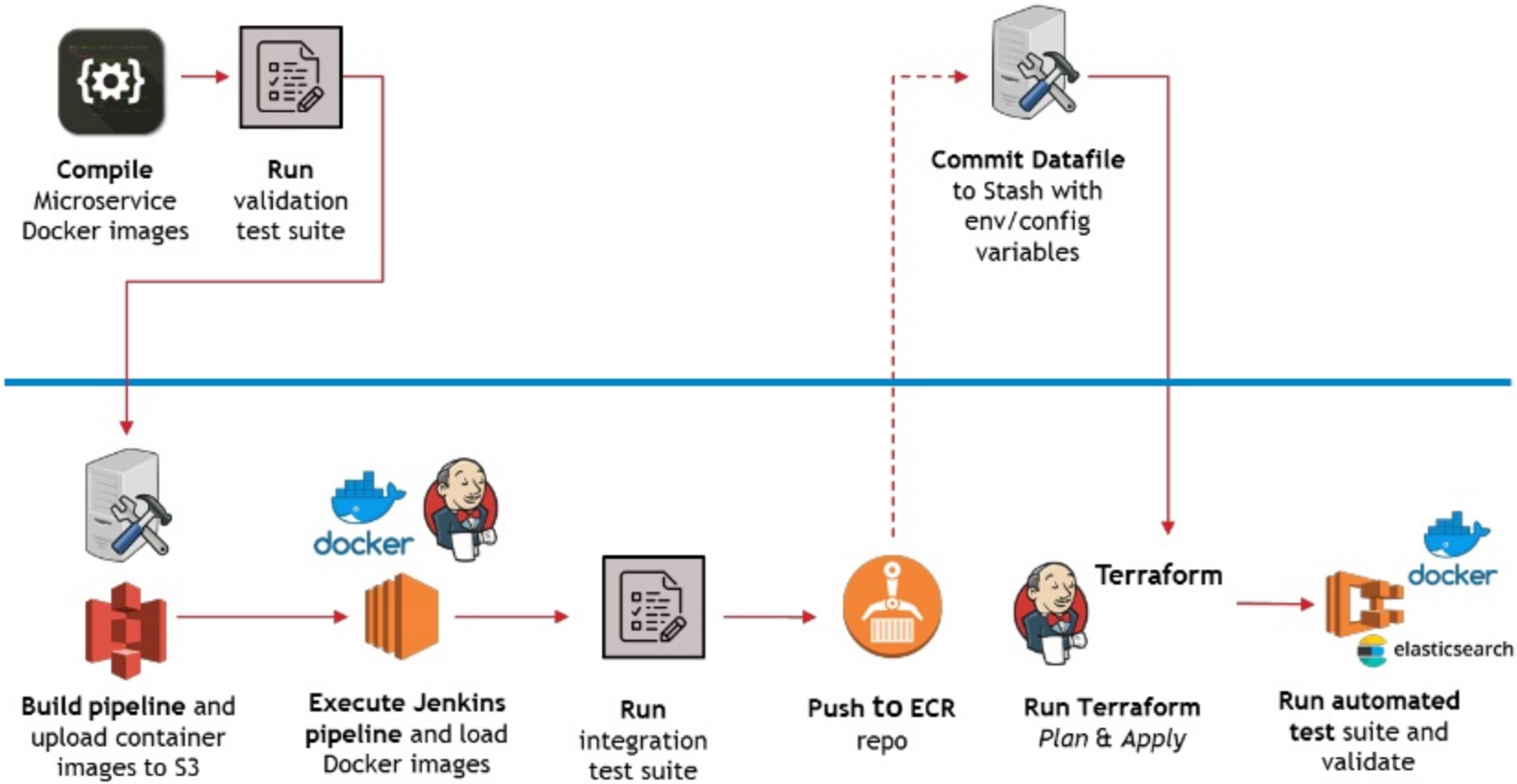


Security

- **Container security**
  - Container isolation through IAM policies and security groups
  - Reduced container-to-container to communication (Reduce attack footprint).

- **ECS instance security**
  - Automated AMI factory → start with ECS optimized AMIs→ McD hardened gold AMIs→ Project specific AMIs
  - AMI factory pipeline listens to a SNS topic for obtaining updated AMIs

DevOps/CI&CD


Compile
Microservice
Docker images
Run
validation
test suite
Commit Datafile
to Stash with
env/config
variables
docker
Terraform
docker
elasticsearch
Build pipeline and
upload container
images to S3
Execute Jenkins
pipeline and load
Docker images
Tag Docker images
with ECR repo
Run
integration
test suite
Push to ECR
repo
Run Terraform
Plan & Apply
Deploy
containers to
ECS
Run automated
test suite and
validate
Monitor

Monitoring

- NewRelic agents configured to monitor the ECS instances, Containers and AWS PaaS components

- ELK stack configured for service logging and analytics

Major Technical Challenges

- "Out of memory error" due to containers not having access to *cgroup* file systems to get memory limits
  - **Solution**: Incorporated a new filesystem(lxcfs) to virtualized *cgroup* and virtualized views of **/proc** files
- Docker containers are not honoring the ECS instance routing rules
  - **Solution:** Custom implementation of a Docker bridge


Final Thoughts
AWS re:Invent
aws

Final Thoughts and Key Takeaways

- A thought out **microservice architecture** is key for **scalability, reliability, and containerization.**
- **Massive scale** achievable (**north of 20k TPS under 100ms**) in a controlled manner **using auto-scale policies and task placement strategies.**
- **Moving to containers** simplified our development and deployment models and in turn provided **quicker dev/test iterations.**
- ECS out of the box integration and deployment models further **simplified our DevOps pipeline.**


AWS re:Invent
THANK YOU!
AWS re:Invent
aws