
AWS
re:Invent

SVS209

Introducing container image support for AWS Lambda

Chris Munns

Senior Manager/Principal Developer Advocate, Serverless

AWS

AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Why are we here today?



© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Today's focus:



Today's focus:

Introducing container image support for AWS Lambda!



But we only have 30 minutes this year …



AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

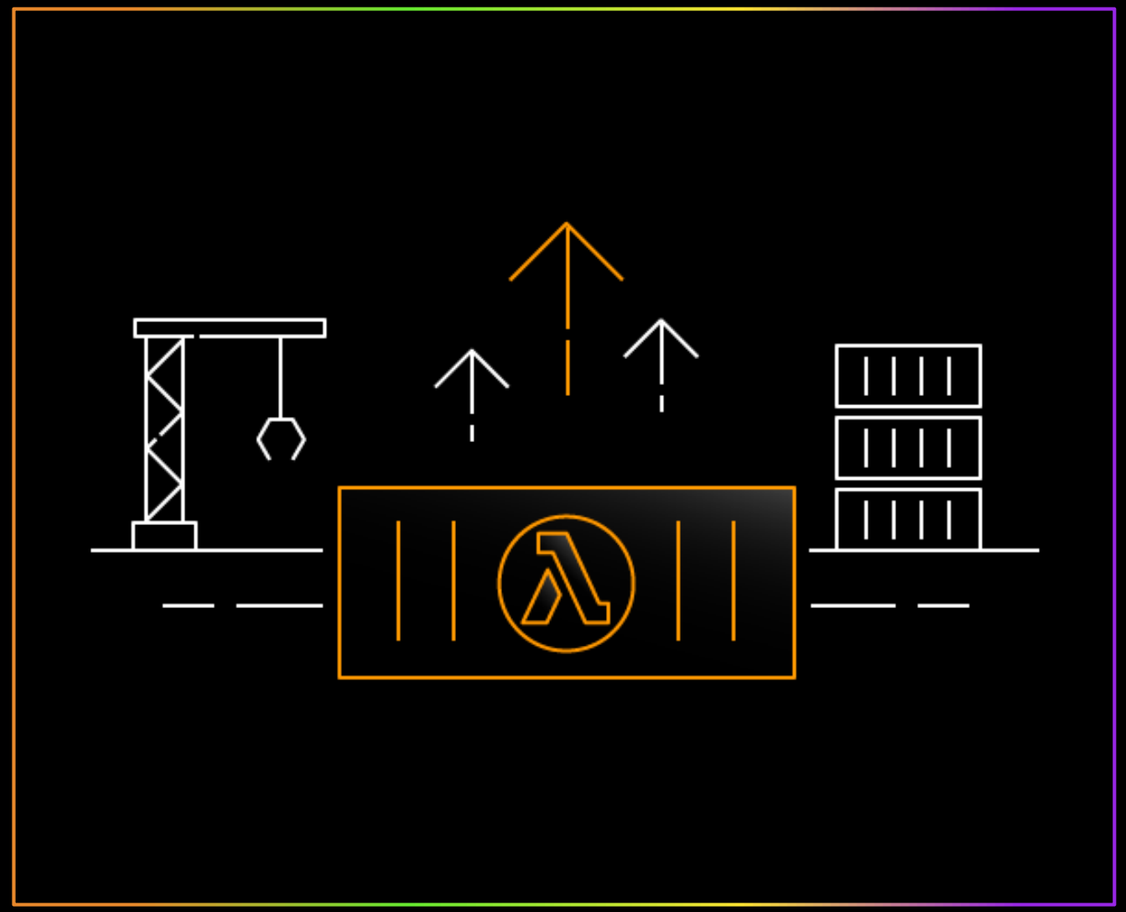
Introducing container image support for Lambda

- What is container image support for Lambda?
- Meeting the RIC and the RIE
- Developer workflow
- What changes and what stays the same?
- Amazon Elastic Container Registry (Amazon ECR)



AWS Lambda container image support

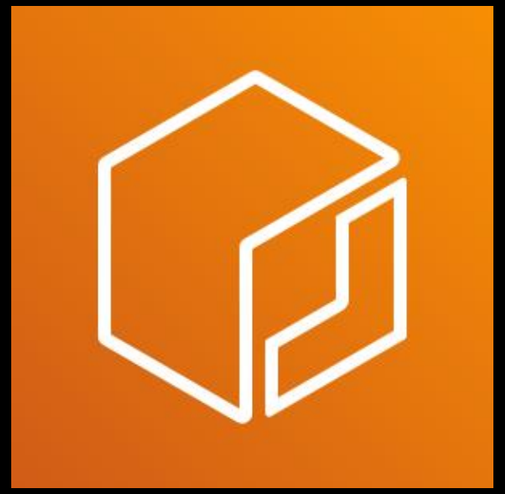
PACKAGE AND DEPLOY FUNCTIONS AS CONTAINER IMAGES



- Easier dependency management and application building with container images
- Use a consistent set of tools for containers and Lambda-based applications
- Deploy large applications with AWS provided or third-party images of up to 10 GB

Why container image support?

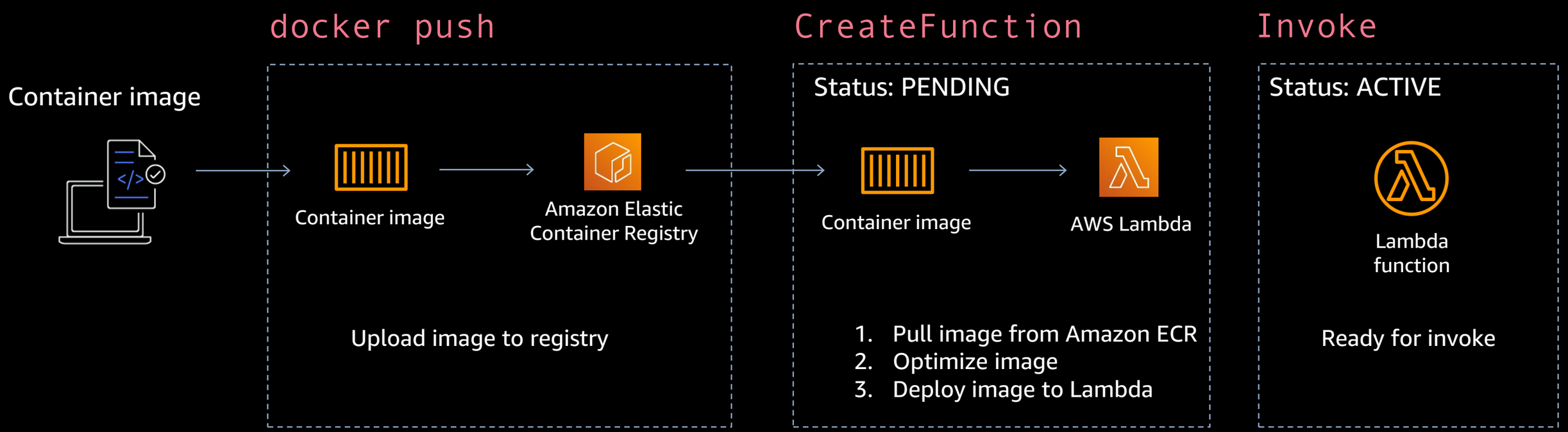
- Easier dependency management with Dockerfile
  - Install native operating system packages
  - Install language-compatible dependencies
- Similar tooling
  - Use the same container registry to store application artifacts (Amazon ECR, Docker Hub)
  - Can utilize the same build and pipeline tools to deploy
  - Tools that can inspect Dockerfiles will work the same
- Larger application artifact up to 10 GB
  - Include larger application dependencies that previously weren’t possible



Lambda container image

Simple developer experience

CREATE FUNCTION AS CONTAINER IMAGE



Looks easy!
But tell me more



© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Working with container images for Lambda

AWS has released base images for every managed runtime for you to get started easily:

- Node.js
- Python
- Java
- .Net
- Ruby
- Go

As well as images for the Runtime API

```
$ cat Dockerfile

FROM public.ecr.aws/lambda/nodejs:12
COPY app.js package*.json ./
RUN npm install
CMD [ "app.lambdaHandler" ]
```

Images available in both Docker Hub and Amazon ECR Public

Other things to know about container images

Lambda supports container images that have manifest files following these formats:

- Docker image manifest V2, schema 2 (used with Docker version 1.10 and newer)
- Open Container Initiative (OCI) Specifications (v1.0.0 and up)

Can use non-Amazon Linux distributions such as Alpine or Debian

Images can be no larger than 10 GB

More here: https://docs.aws.amazon.com/lambda/latest/dg/images-create.html

Working with container images for Lambda

AWS has released base images for every managed runtime for you to get started easily:

- Node.js
- Python
- Java
- .Net
- Ruby
- Go

As well as images for the Runtime API

```
$ cat Dockerfile

FROM public.ecr.aws/lambda/nodejs:12
COPY app.js package*.json ./
RUN npm install
CMD [ "app.lambdaHandler" ]
```

Images available in both Docker Hub and Amazon ECR Public

Working with container images for Lambda

AWS has released base images for every managed runtime for you to get started easily:

- Node.js
- Python
- Java
- .NET
- Ruby
- Go

As well as images for the Runtime API

Images available in both Docker Hub and Amazon ECR Public

But what's this? →

```
$ cat Dockerfile

FROM public.ecr.aws/lambda/nodejs:12
COPY app.js package*.json ./
RUN npm install
CMD [ "app.lambdaHandler" ]
```

Dockerfile for Lambda functions

```
FROM public.ecr.aws/lambda/nodejs:12

COPY app.js package*.json ./

RUN npm install

CMD [ "app.lambdaHandler" ]
```

Use the node.js v12 base image from Amazon ECR Public

Copy my important application files to the image

Run the npm command to install dependencies in the image

When the container runs, use my application's handler

Meeting the RIC and the RIE

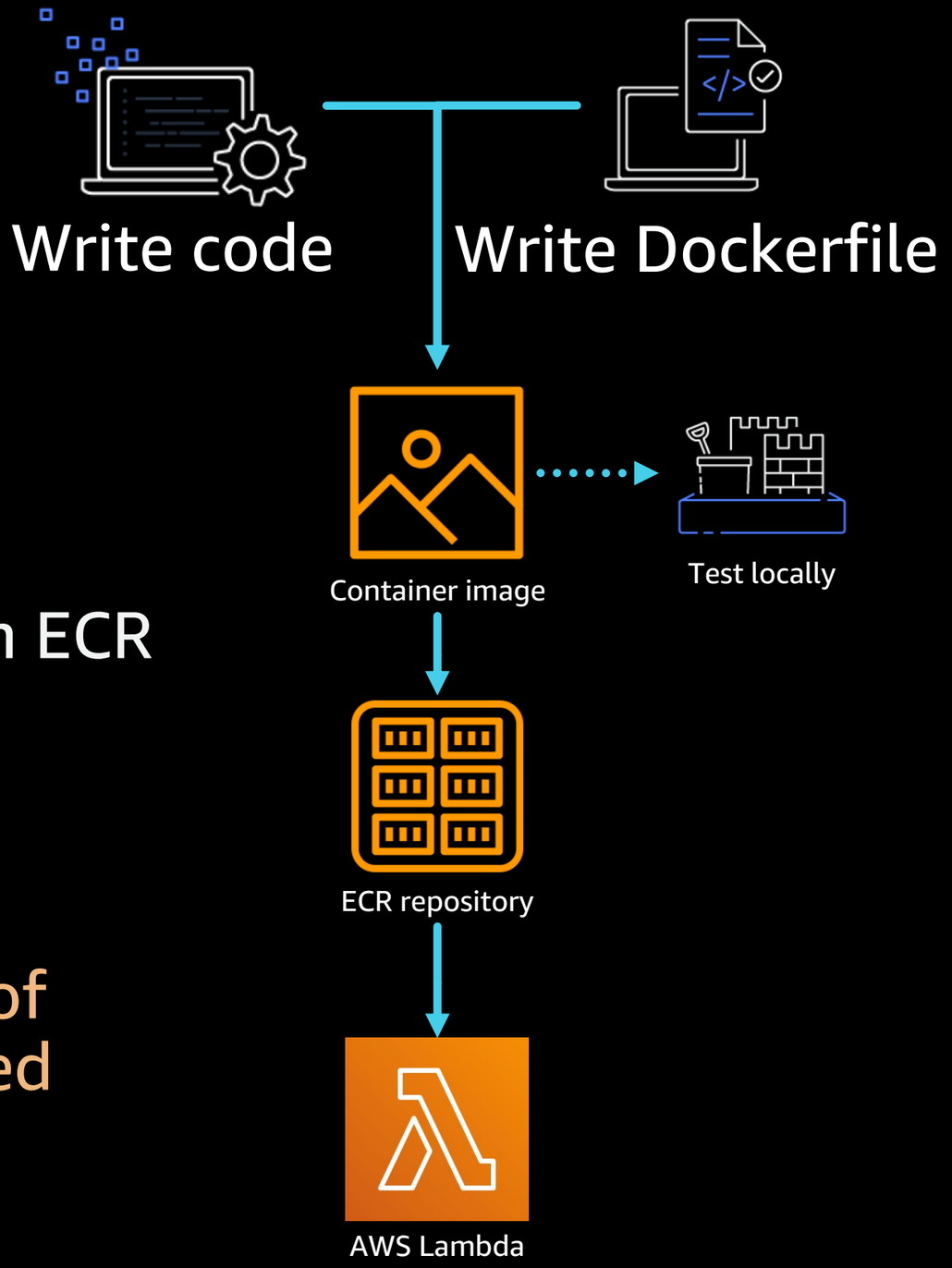
There are two extra components we've released as part of container image support:

- AWS Lambda Runtime Interface Client(s) (RIC)
  - Allows container-based function code to connect to the Runtime API
  - AWS has released these for each managed runtime
  - Open-sourced (for example: https://github.com/aws/aws-lambda-nodejs-runtime-interface-client)
- AWS Lambda Runtime Interface Emulator (RIE)
  - Mimics the Runtime API locally for testing purposes
  - Exposes an HTTP interface that converts requests to JSON for the RIC
  - Also open-sourced: https://github.com/aws/aws-lambda-runtime-interface-emulator/

Build and deploy flow

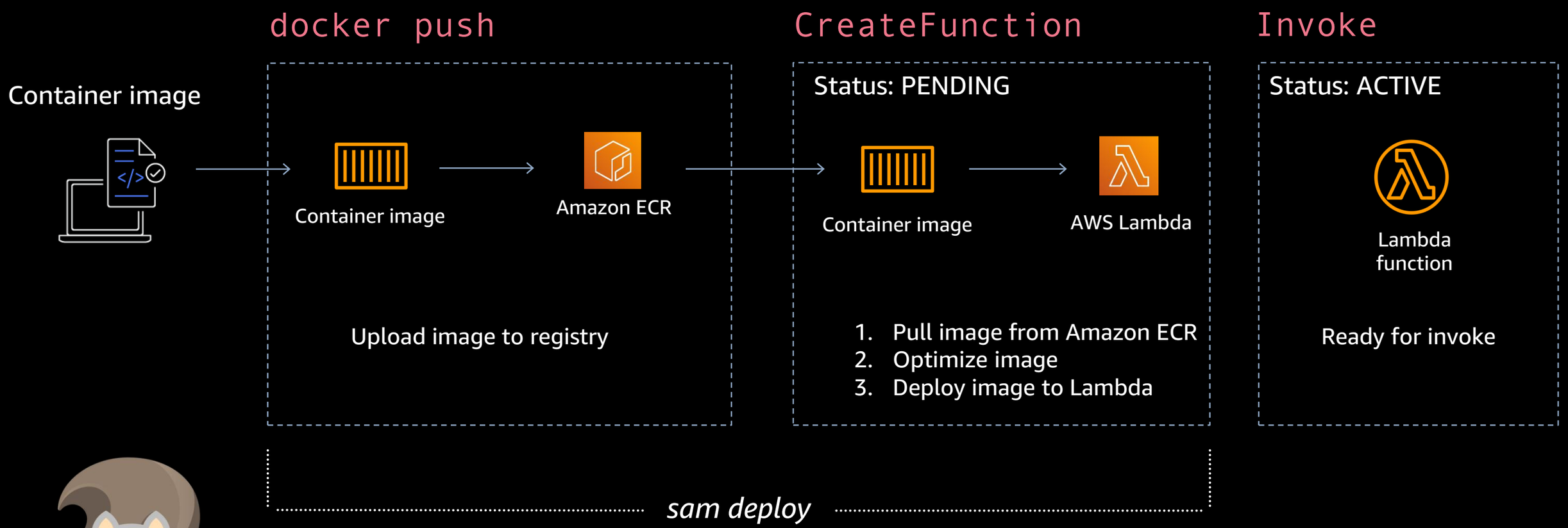
1. Create / update application
2. Create / update Dockerfile
3. Run `docker build`
    1. Optional – local test of function capability
4. Push created container image to Amazon ECR
5. Create / update Lambda function

On create / update Lambda will cache the image in the service to speed up cold start of functions during execution or for provisioned concurrency spin-up



AWS SAM developer experience

CREATE FUNCTION AS CONTAINER IMAGE





AWS SAM will package the container, push it to a repo, and create or update the Lambda function with a single command

What changes?
What stays the same?



© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Changes from container image support

What changes?

- Larger artifact now possible
- Require RIC to talk to Runtime API
- No auto-updates to managed runtimes without a redeploy

What stays the same?

- Invoke model
- Storage options
- Logging / metrics / AWS X-Ray
- Other service limits
- Billing

Anatomy of a Lambda function

```
let response;

exports.handler = async (event, context) => {
  try {
    response = {
      'statusCode': 200,
      'body': JSON.stringify({
        message: 'hello world’,
      })
    }
  } catch (err) {
    console.log(err);
    return err;
  }

  return response
};
```

Handler function

Function to be executed upon invocation

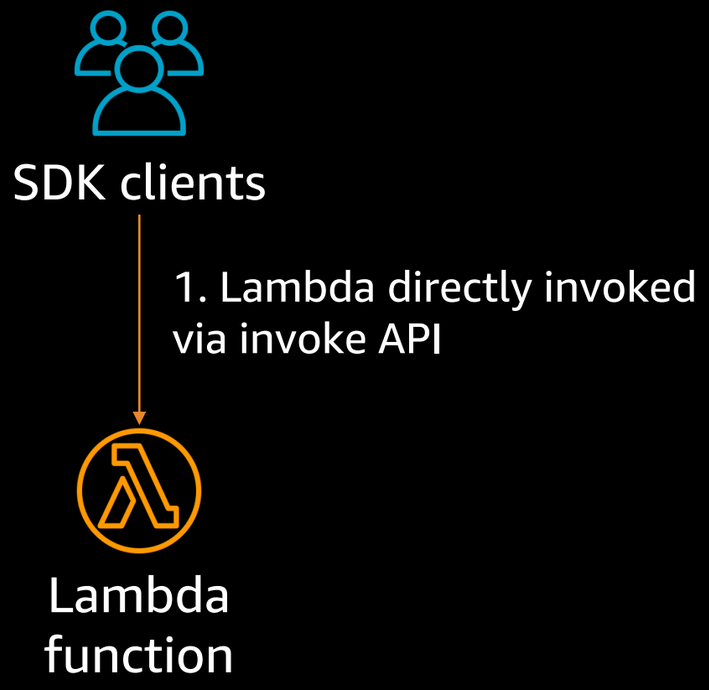
Event object

Data sent during Lambda function invocation

Context object

Methods available to interact with runtime information (request ID, log group, more)

Lambda API



API provided by the Lambda service

Used by all other services that invoke Lambda across all models

Supports sync and async

Can pass any event payload structure you want

Client included in every SDK

Lambda execution model

Synchronous (push)



Amazon API Gateway

/order



Lambda function

Asynchronous (event)



Amazon SNS



Amazon S3

reqs



Lambda function

Stream (Poll-based)



Amazon DynamoDB



Amazon Kinesis

changes

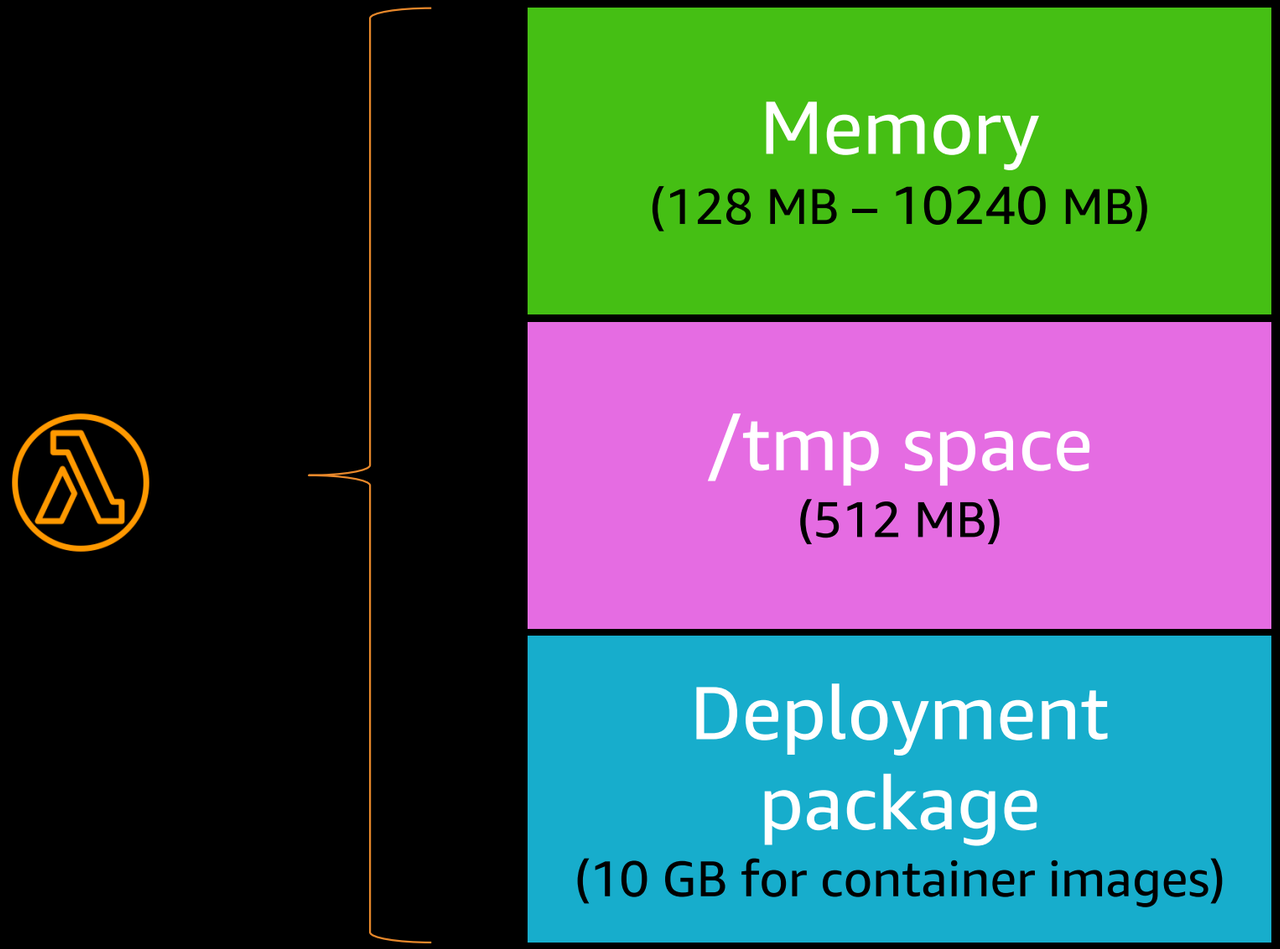


AWS Lambda service

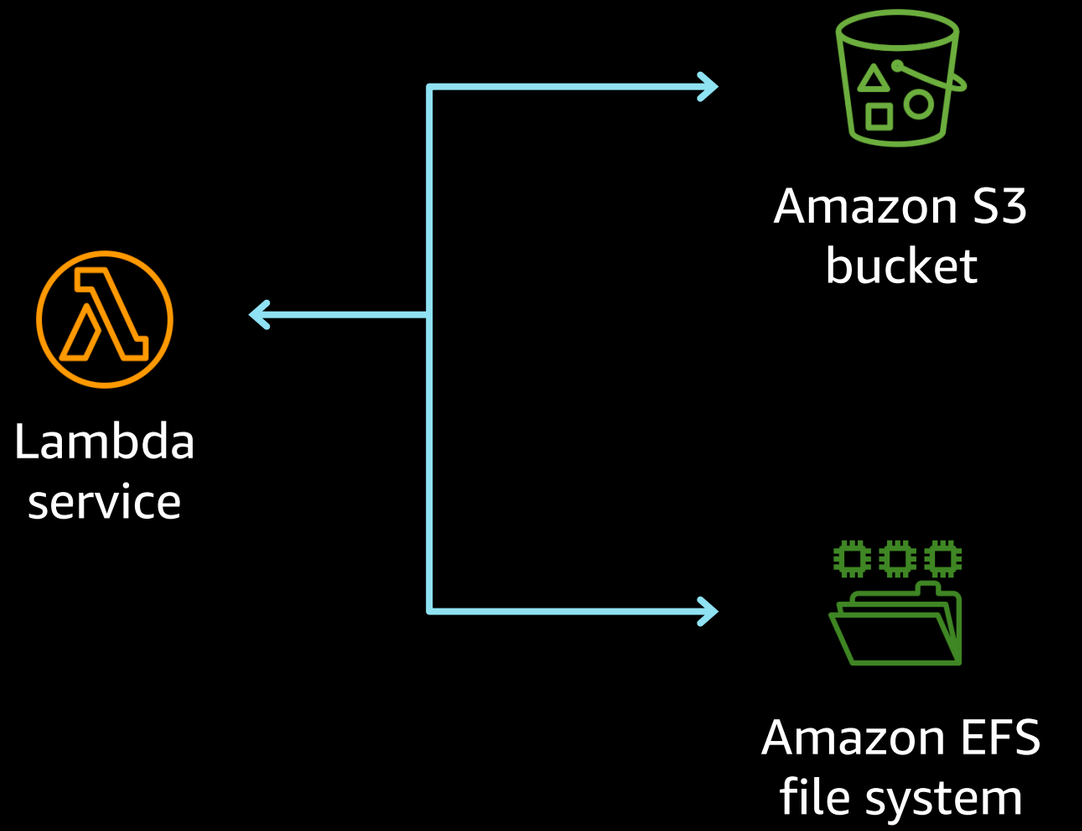


function

Storage in Lambda



Lambda external storage options



AWS Lambda: Larger Lambda functions

Developers can now configure Lambda functions for:

10 GB in memory with up to

6 vCPUs proportional to memory configuration

Build compute-intensive workloads:

- Machine learning, genomics, gaming, HPC applications

Build memory-intensive workloads:

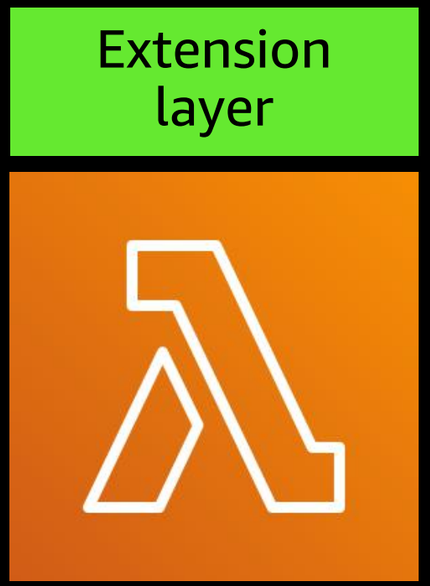
- Batch, ETL, analytics, media processing

Configured memory (MB)	Allocated cores
128 – 1769	1
1770 – 3538	2
3539 – 5307	3
5308 – 7076	4
7077 – 8845	5
8846 – 10240	6

AWS Lambda Extensions

In preview

A new way for developer tools and third-party tools to integrate deeply into the Lambda environment



- Capturing diagnostics before, during, and after function invocation
- Instrumenting code without changing it
- Fetch configuration settings or secrets before function invocation
- Detect and alert on function activity through hardened security agents

https://s12d.com/extensions-intro

Importing Lambda Extensions from ECR

```
...
FROM <accountid>.dkr.ecr.<region>.amazonaws.com/sample-extension:latest AS some-extension
WORKDIR /opt/extensions
COPY --from=some-extension some-extension .
...
```

Importing existing Lambda layers

```
FROM alpine:latest AS layer-grabber

WORKDIR /layer
RUN apk add aws-cli curl
#
ARG AWS_DEFAULT_REGION=${AWS_DEFAULT_REGION:-"us-east-1"}
ARG AWS_ACCESS_KEY_ID=${AWS_ACCESS_KEY_ID:-""}
ARG AWS_SECRET_ACCESS_KEY=${AWS_SECRET_ACCESS_KEY:-""}

ENV AWS_DEFAULT_REGION=${AWS_DEFAULT_REGION}
ENV AWS_ACCESS_KEY_ID=${AWS_ACCESS_KEY_ID}
ENV AWS_SECRET_ACCESS_KEY=${AWS_SECRET_ACCESS_KEY}

RUN curl $(aws lambda get-layer-version-by-arn --arn <layer-arn> --query 'Content.Location' --output
text) --output layer.zip
RUN unzip layer.zip
RUN rm layer.zip

...
WORKDIR /opt
COPY --from=layer-grabber /layer/python/ .
```

AWS Lambda: 1 ms billing



- Reduced billing granularity from 100 ms to 1 ms
- Affects all Lambda functions
- Automatic – no action needed
- Starts with December 2020 billing cycle

Container image security

Container images are stored in Amazon Elastic Container Registry (ECR)

Amazon ECR provides several mechanisms for security and controlling access to container images:

- ECR repositories are private by default
- Support fine-grained AWS Identity and Access Management (IAM) permissions
- AWS CloudTrail logging of actions on repositories
- Encryption at rest and in transport to Lambda
- Cross-account sharing controls
- Can replicate images across Regions
- Integrated vulnerability scanning



Amazon Elastic Container Registry

AWS Consulting Partners are here to help







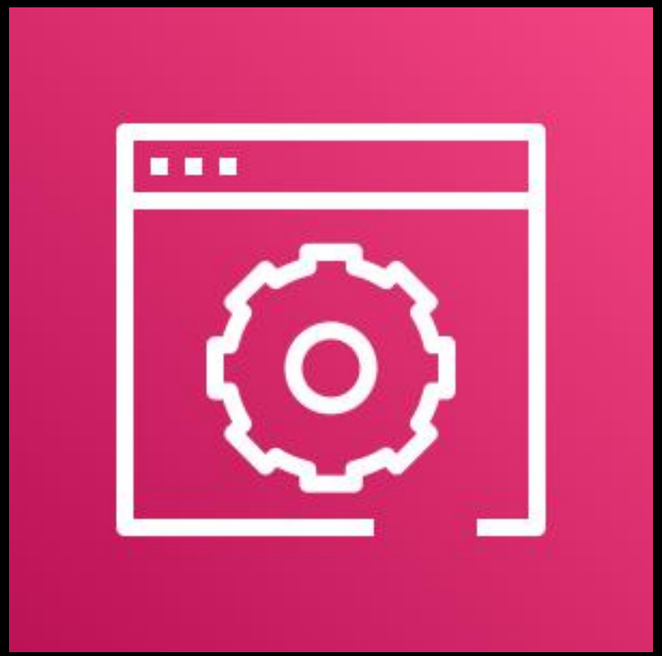




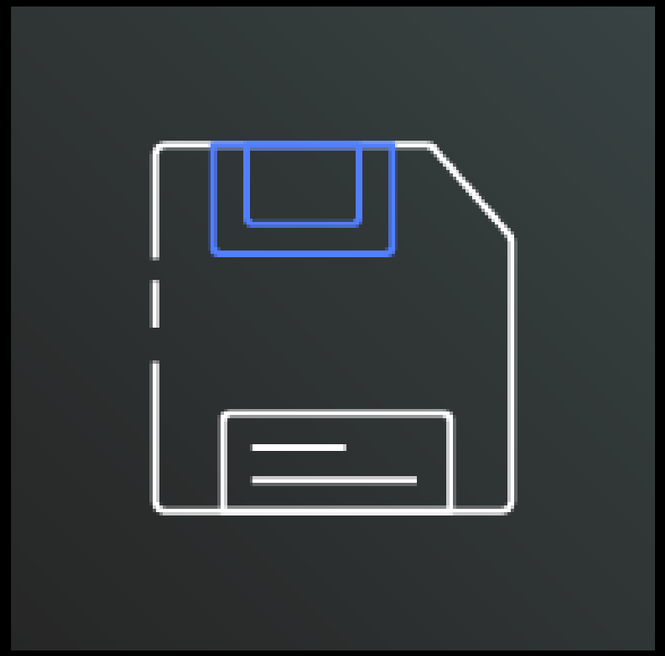


This is not a complete list. To view all AWS Lambda Delivery Partners, visit https://aws.amazon.com/lambda/partners

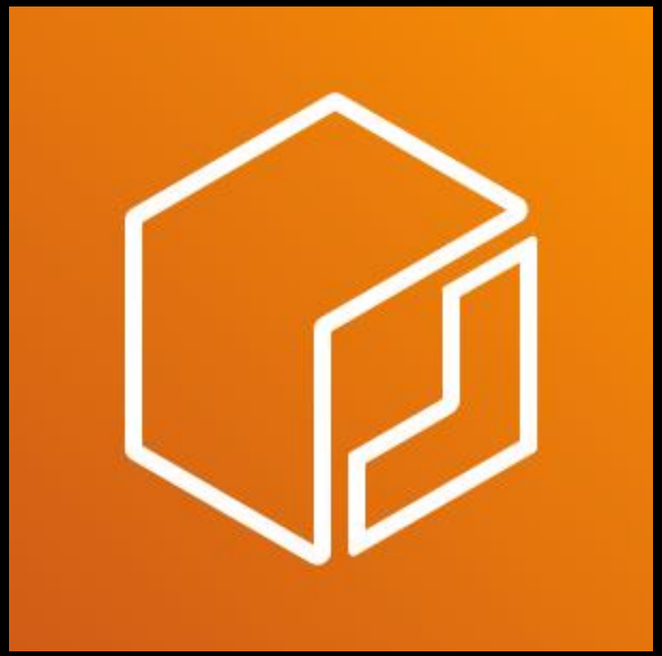
Create and package your functions



Lambda
console



Lambda
zip file



Lambda
container image

FIN / ACK

Container image support can simplify the way you build Lambda-based applications:

- Easier dependency management
- Unified tooling with container-based workloads
- Larger application artifacts

Same benefits of Lambda as a compute service and simplified developer experience with AWS SAM

So much more here:

https://s12d.com/svs209

Thank you!

Chris Munns

munns@amazon.com
@chrismunns

AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.



Please complete the session survey

AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.