AWS Developer - An Introduction to AWS Lambda

UNDERSTANDING SERVERLESS FUNCTIONS

Fernando Medina Corey
SUPPORT ENGINEER

@fmcorey www.fmcorey.com

Outline

Evolution of serverless functions

- Serverless vs. traditional architecture
- What are serverless functions
- Benefits & drawbacks

Serverless function providers

- AWS, competitors, and niche players

Demo overviews

- What are we building?
- How are we building it?

Evolution of Serverless Functions

Serverless vs. Traditional Architecture

1950s-1970s

IBM and others

Mainframes

2005

VMware

OS Virtualization

2006

AWS EC2/S3

The Cloud

1977

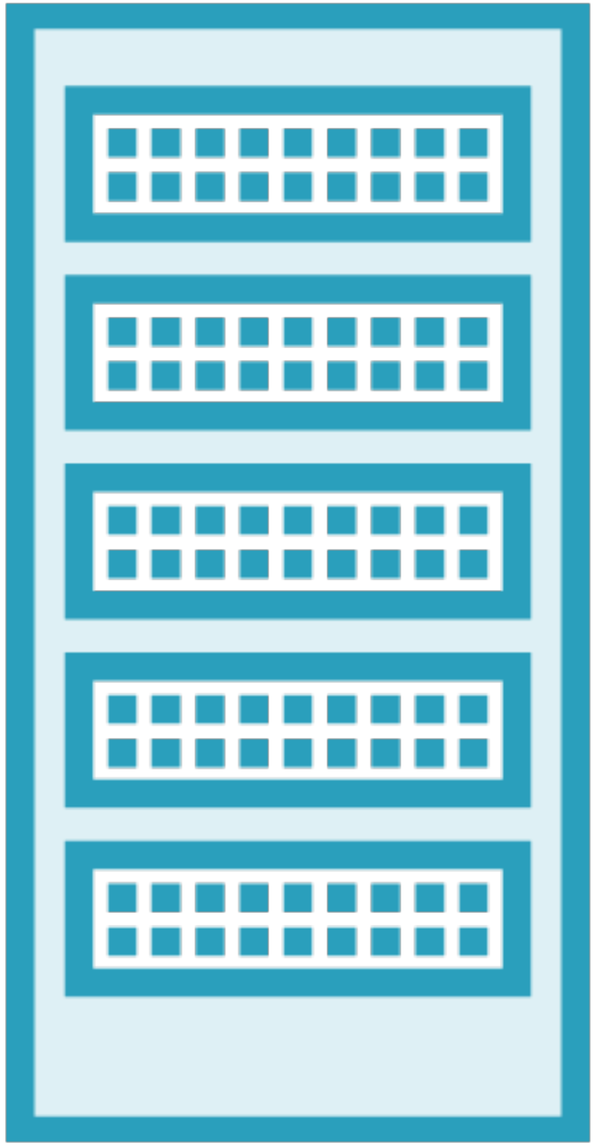
Apple and others

Personal Computing

2014

AWS Lambda 2014

Serverless



Mainframes

- **Large space requirement**
- **Installation**
- **Maintenance**
- **Cost**
- **Inflexibility**

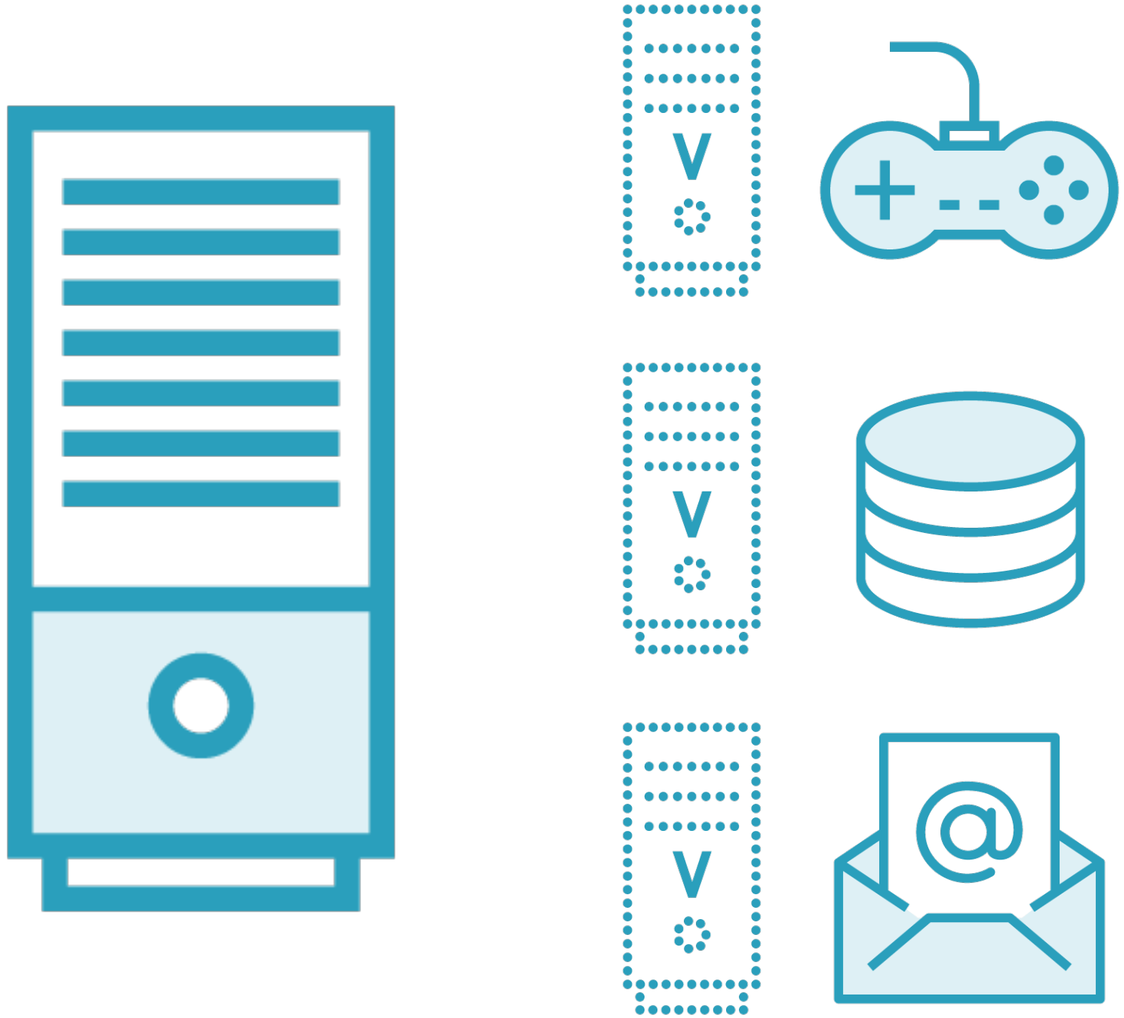
Personal Computing / Minicomputers

Lower barriers to entry

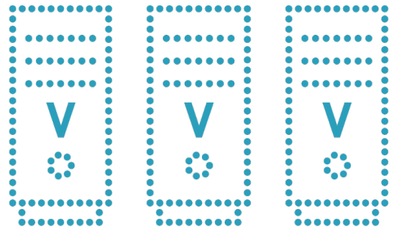
Reduced cost

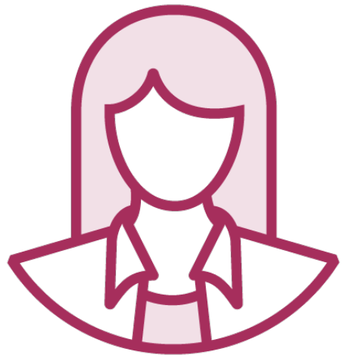
Increased distribution

Virtualization and Hypervisors



The Cloud - Amazon EC2





Serverless Functions

Event driven

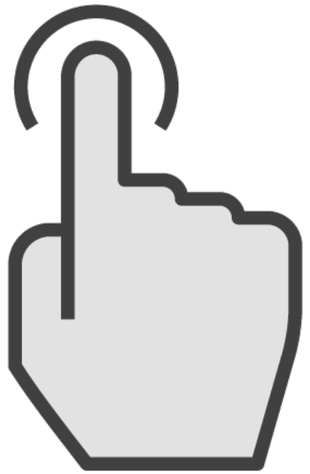
Code focused

Managed machines

Event Examples

File uploads

Scheduled times



API requests

Serverless Benefits and Drawbacks

Benefits

Cost and utilization

Managed machines

Service integrations

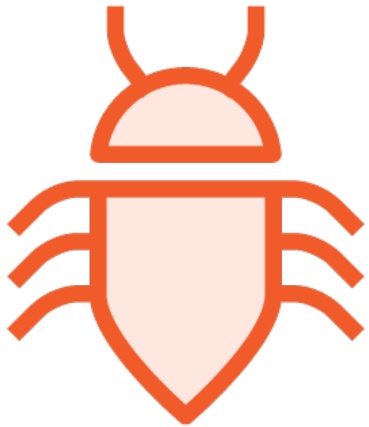
Scaling

Drawbacks

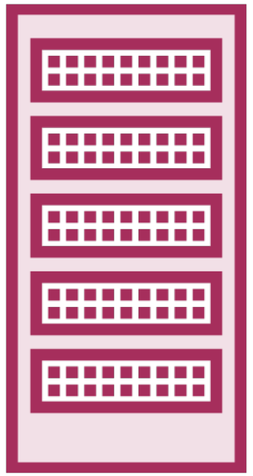
Debugging

Control

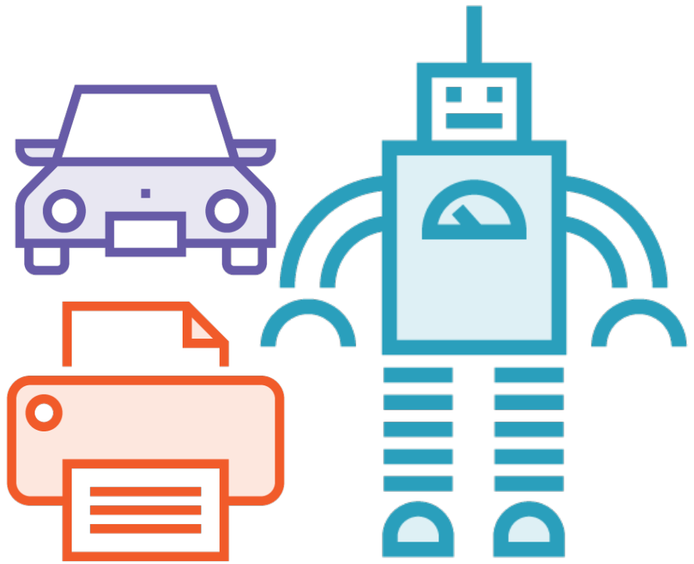
Cutting-edge quirks



Why Learn Lambda?



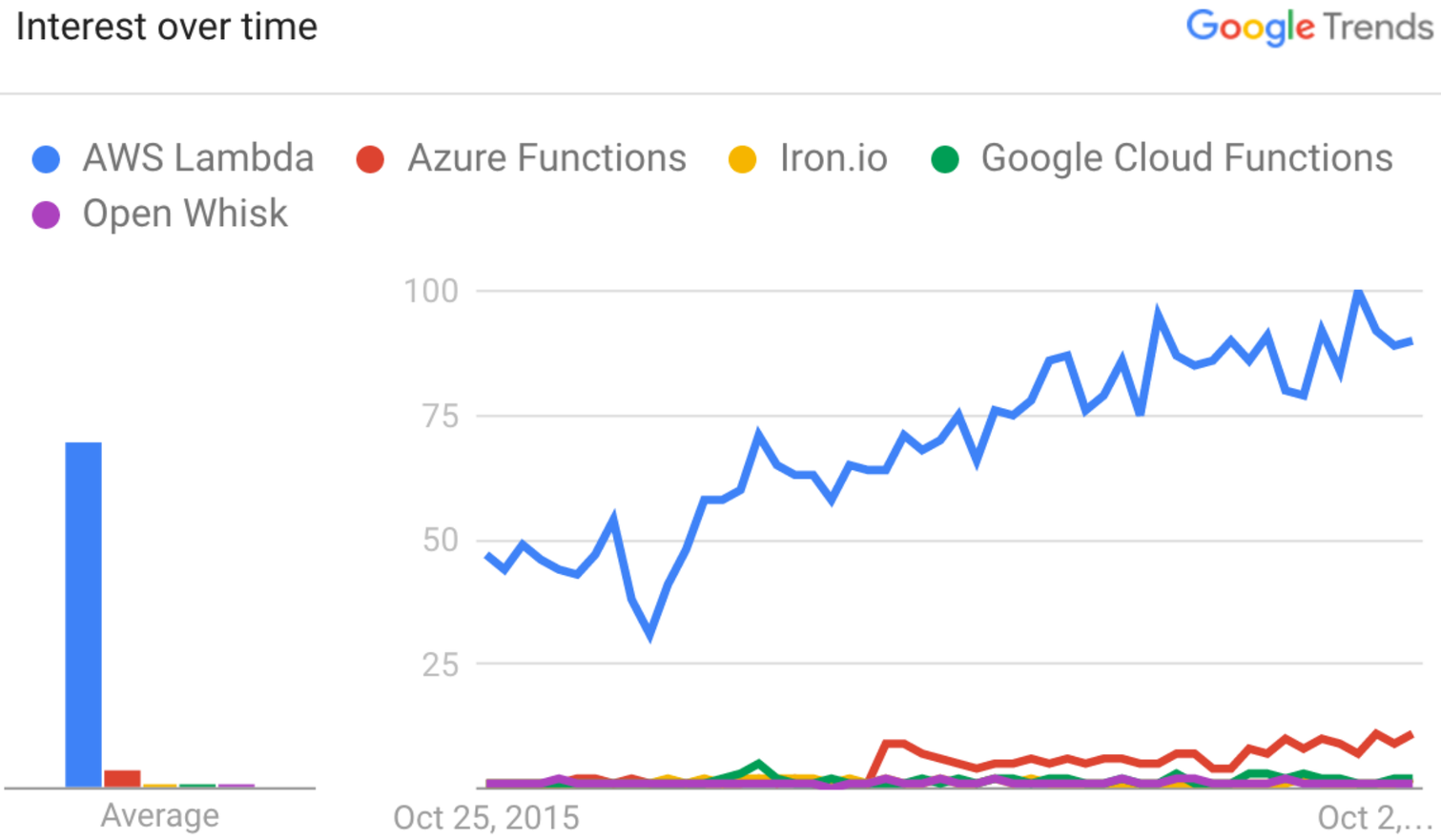
Managed infrastructure



Internet of Things

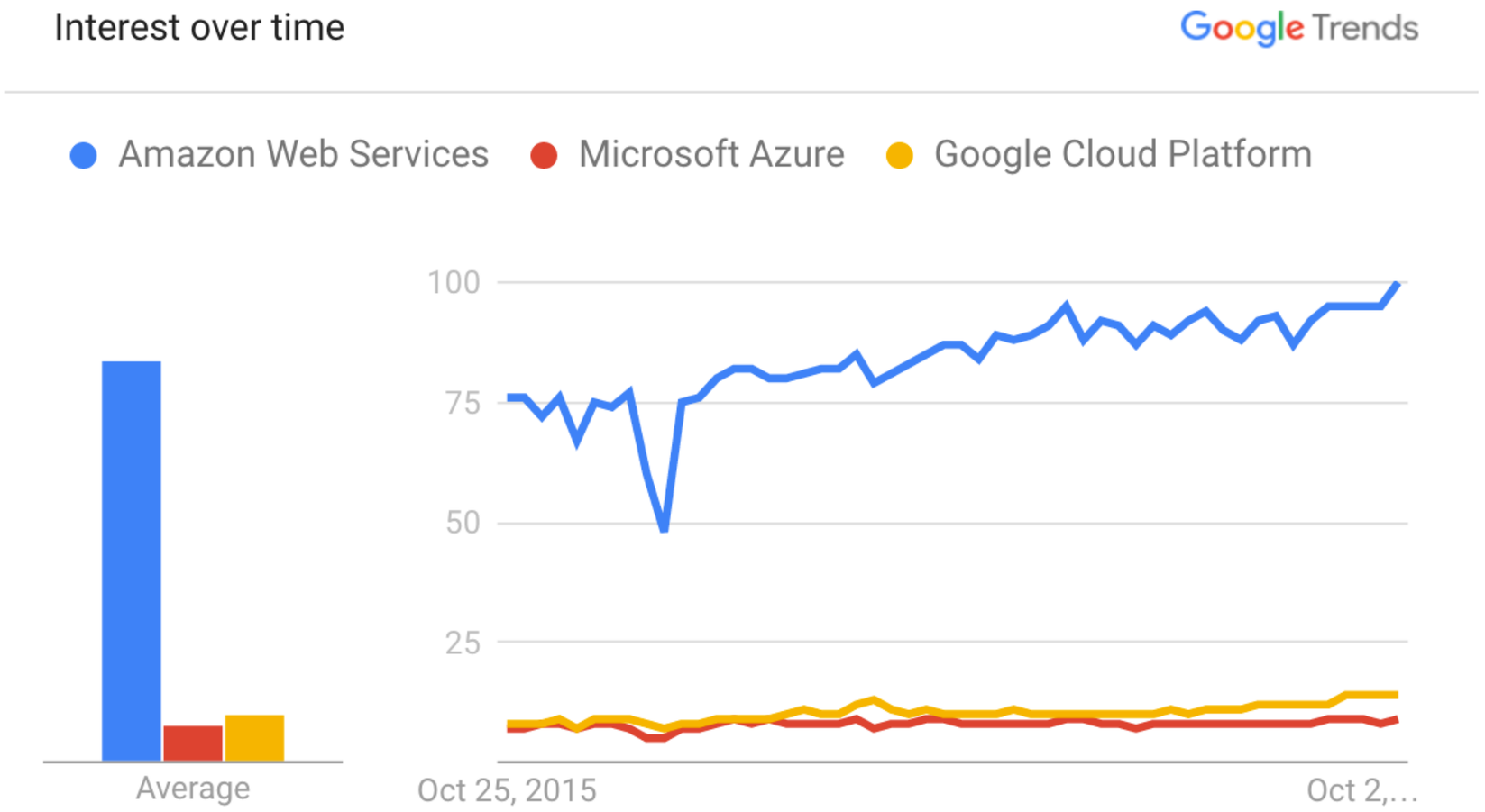
Growing relevance

Lambda's Growing Relevance

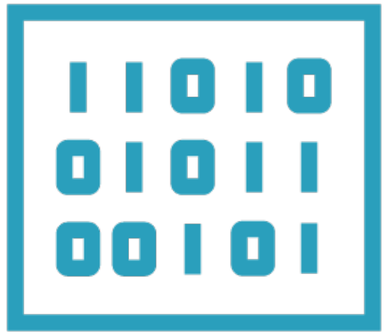


Data source: Google Trends (www.google.com/trends)

AWS and Competitors



How Is Lambda Used?



Stream data processing

Easy & scaleable APIs

Photo processing

Web applications

Serverless Function Providers

Prominent Serverless Function Providers

Market leaders:
AWS, Microsoft Azure

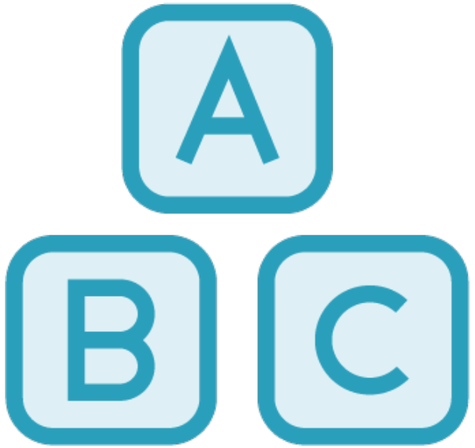
Other players:
IBM, Google, iron.io

Market Leader Comparison

AWS Lambda	Azure Functions
Python, Node.js, Java, Node shims of other languages	C#, Node.js, Python, F#, PHP, batch, bash, Java, or any executable
Built-in versioning	No built-in versioning
Closed source runtime	Open source runtime
HTTP endpoints via API Gateway	Automatic HTTP endpoints
5 minute running time limit	No running time limit
100 concurrent functions (soft limit)	10 concurrent instances

Niche Providers

Iron.io



Google Cloud Functions

Open Whisk (IBM)

Demo Overviews

Woof Garden - Our Demo Client

Needs:

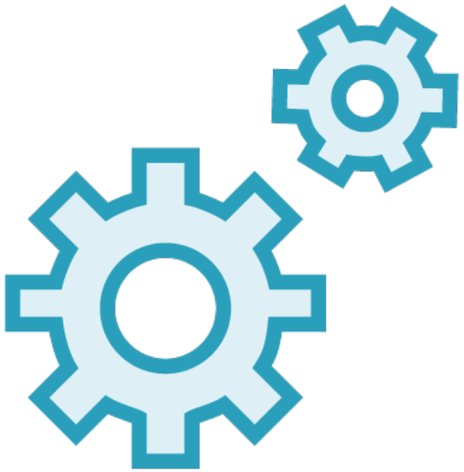
Website uptime monitoring

Social media automation

Custom business reminders

Our Three Lambda Projects

Simple scheduled events

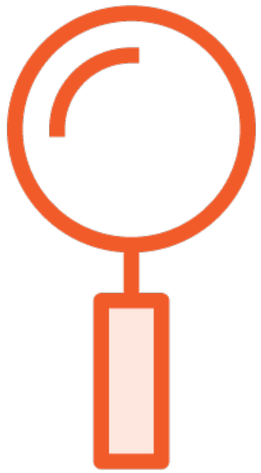


3rd party APIs

Business logic and AWS SES

Lambda Canary

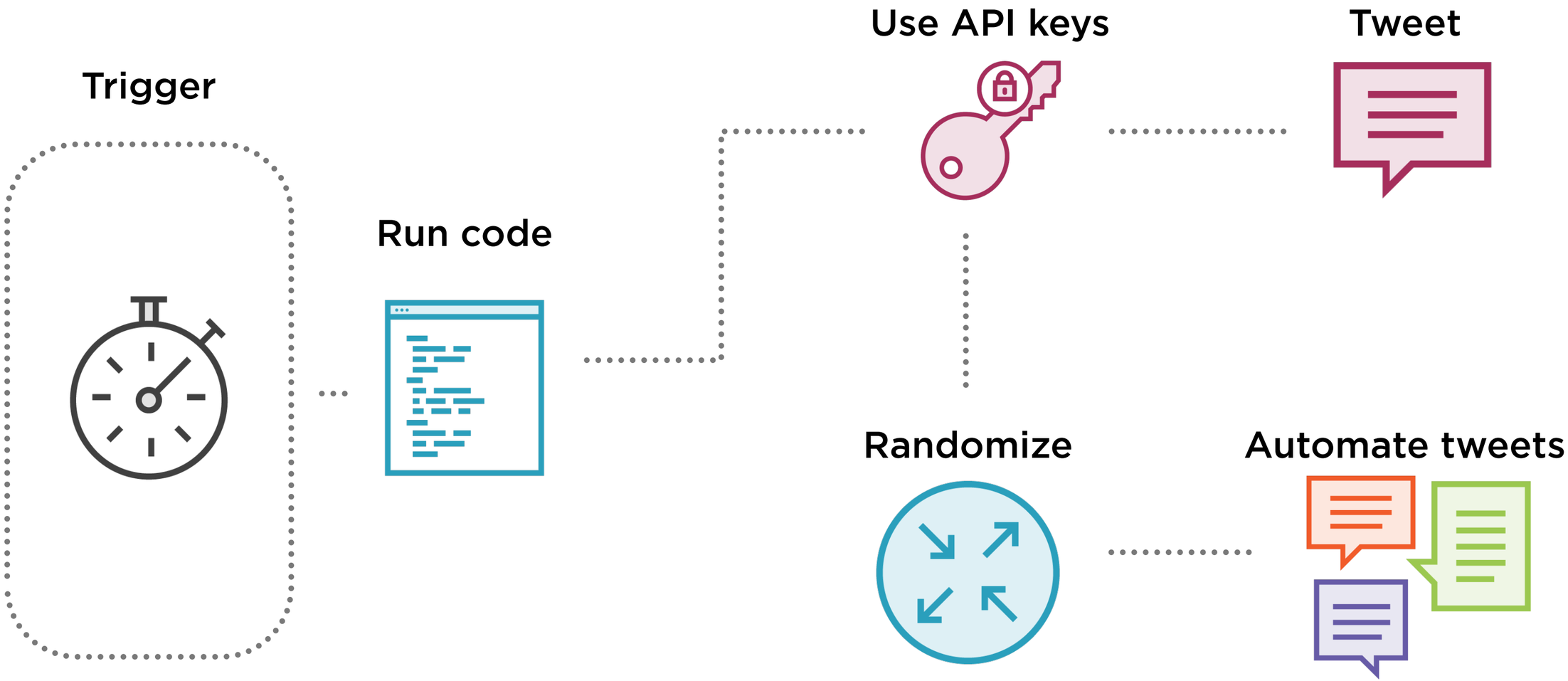
Set run interval



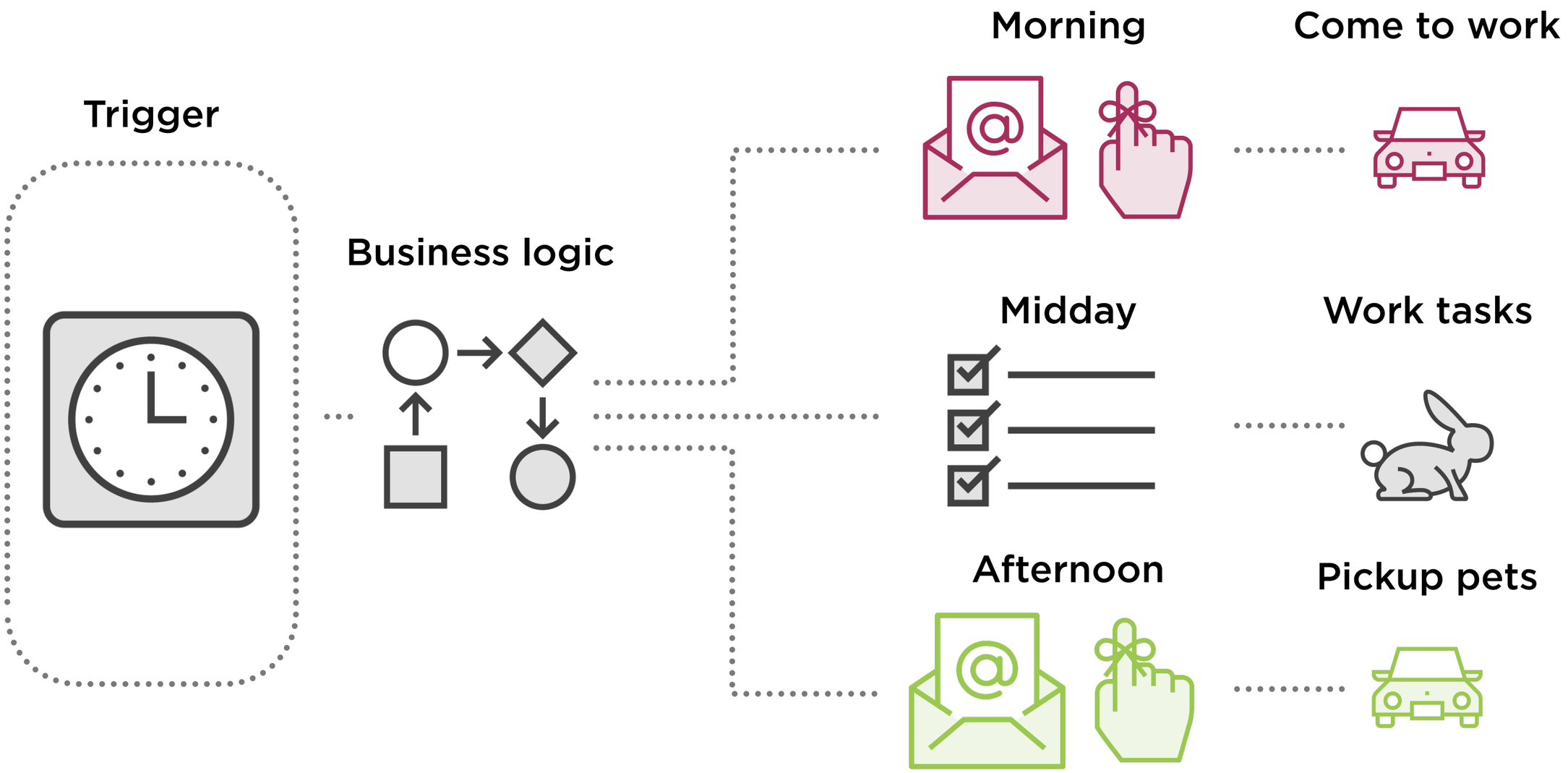
Function reviews website

Website status recorded

Twitter Bot


Trigger
Run code
Use API keys
Tweet
Randomize
Automate tweets

Workflow Automation



Summary

What we covered

- Context of Serverless functions
- Current landscape
- When to (not) use Serverless
- Introduced our projects

What's next?

- AWS crash course

Outline

Creating an AWS free tier account

- What is free tier?
- Walkthrough signup process
- How to stick to 100% free services

Navigating the AWS console

- Service areas & menus
- Services we will be using

Identity & Access Management (IAM)

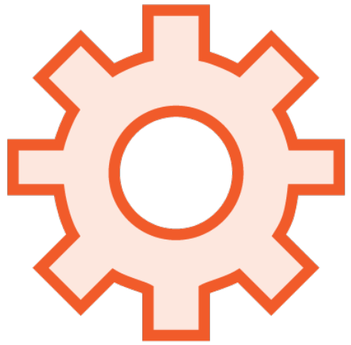
- The basics of roles and policies
- IAM setup with the policy generator

AWS Free Tier - 12 Months of Free

Amazon EC2

Dynamo DB

Simple Email Service



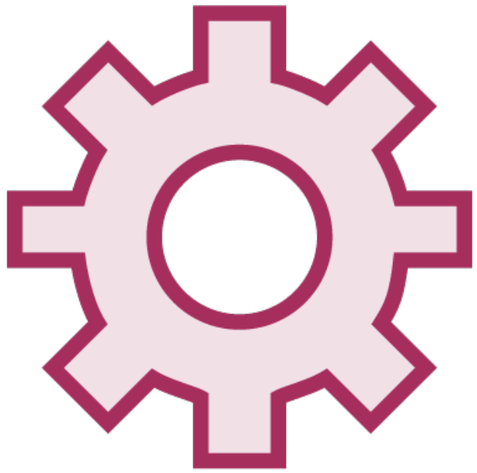
Lambda

Relational Database Service

Simple Storage Service

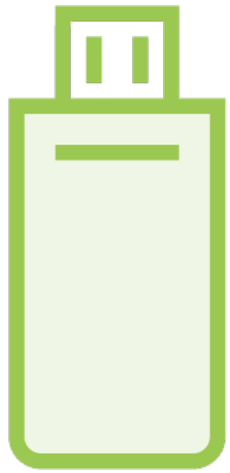
Free Tier Examples

EC2 - 750 Hours



Lambda
Functions - 1MM

Dynamo - 25GB
RDS - 20GB



AWS S3 - 5GB

Building with Free Tier

Create static sites on S3

Database powered applications with EC2 and RDS

Scaleable APIs with Lambda and API Gateway

Demo

Free tier signup

- Walkthrough
- Finding service-specific information
- Avoiding unwanted expenses

AWS console

- Service areas
- Configuring our workspace
- Finding services we'll use

Identity and Access Management - Overview

A Few Key Concepts

Users

Groups

Roles

Users, Groups, Roles, and Policies

Users, Groups, Roles, and Policies

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "*",
      "Resource": "*"
    }
  ]
}
```

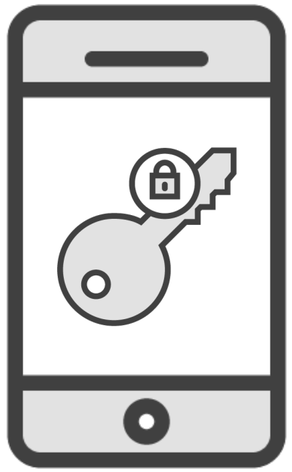
Sample IAM Policy

- JSON
- Tracks policy versions

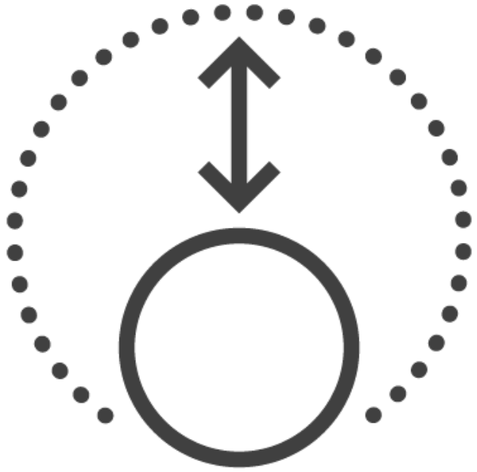
- Allow or deny access
- What actions can be taken?
- On what AWS resources

IAM Best Practices

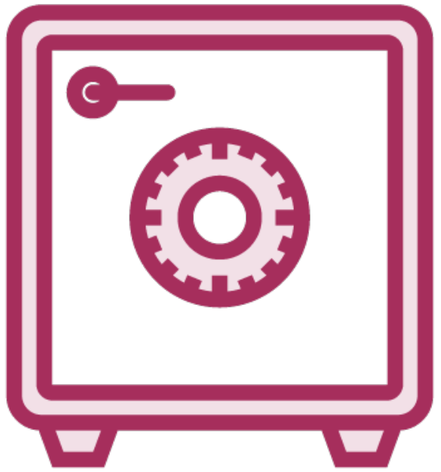
Strong password



Two factor authentication



Principle of Least Privilege



Root account caution!

Demo

AWS IAM Demo

- AWS Managed Policies
- AWS Policy Generator
- AWS IAM roles

Summary

What we've done

- Building with Free Tier
- Understand AWS services
- Created IAM policies

What's next?

- Our first Lambda project

Outline

Preparing

- Determine functionality
- Gather credentials & libraries
- Prepare configuration file

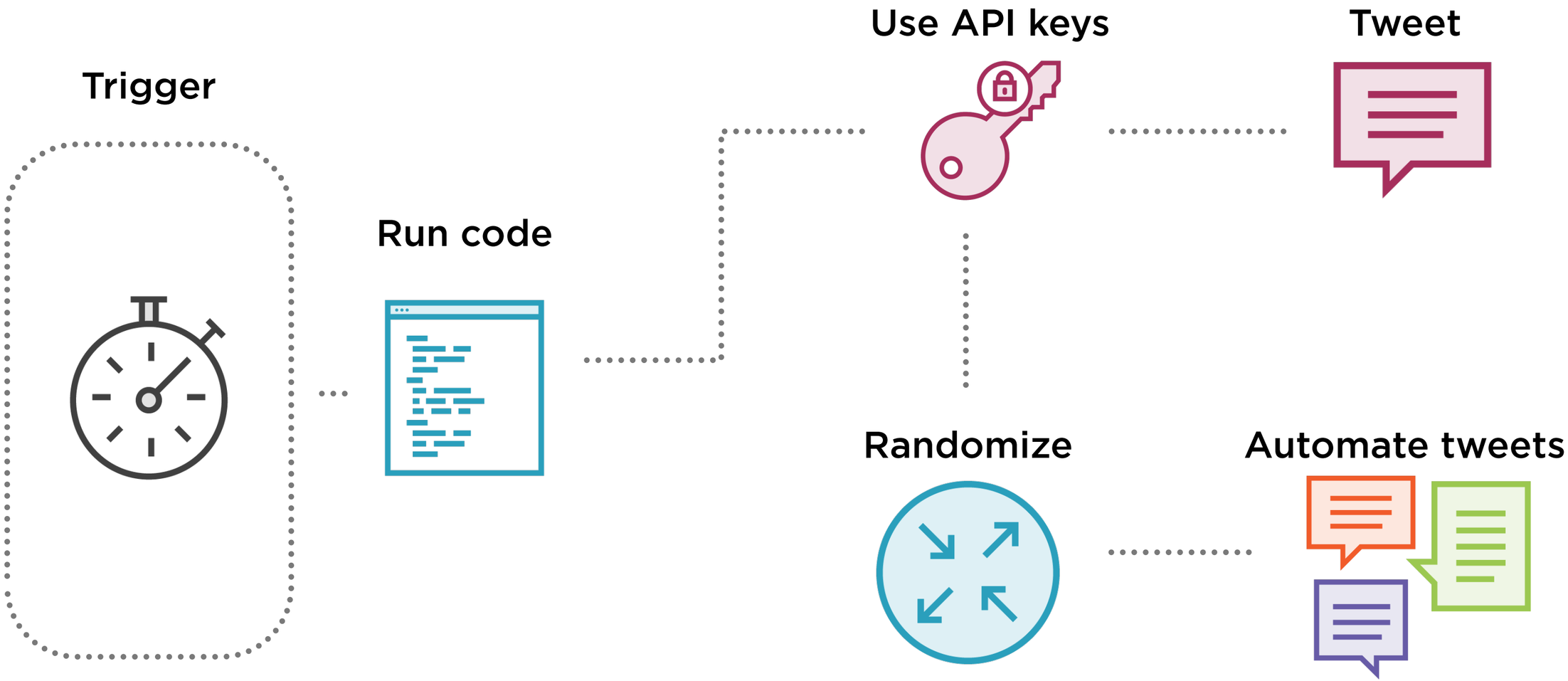
Write and Test Our Function

- Modify a bot skeleton
- Test locally

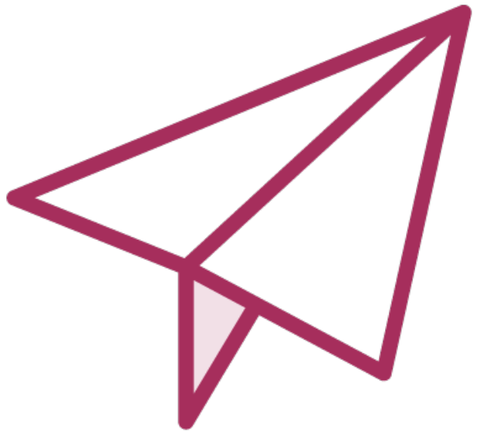
Deploying Our Bot

- Creating a function package
- Configuration and deployment

Determine Functionality - Twitter Bot



Credentials and Libraries

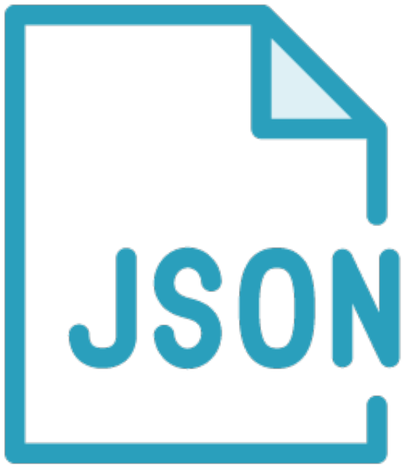


Sparrow Twitter bot shell

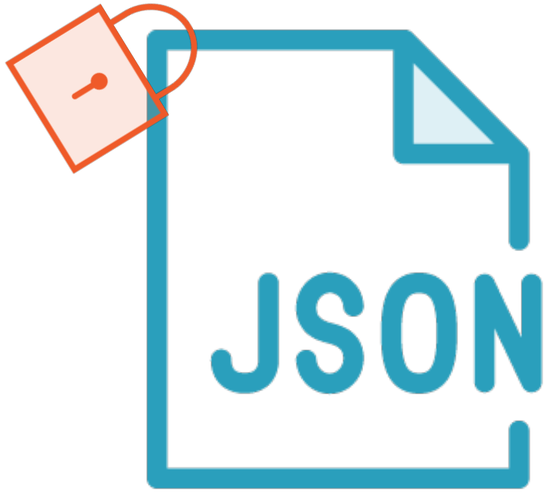
Twython

Twitter API keys

API Key Management Options

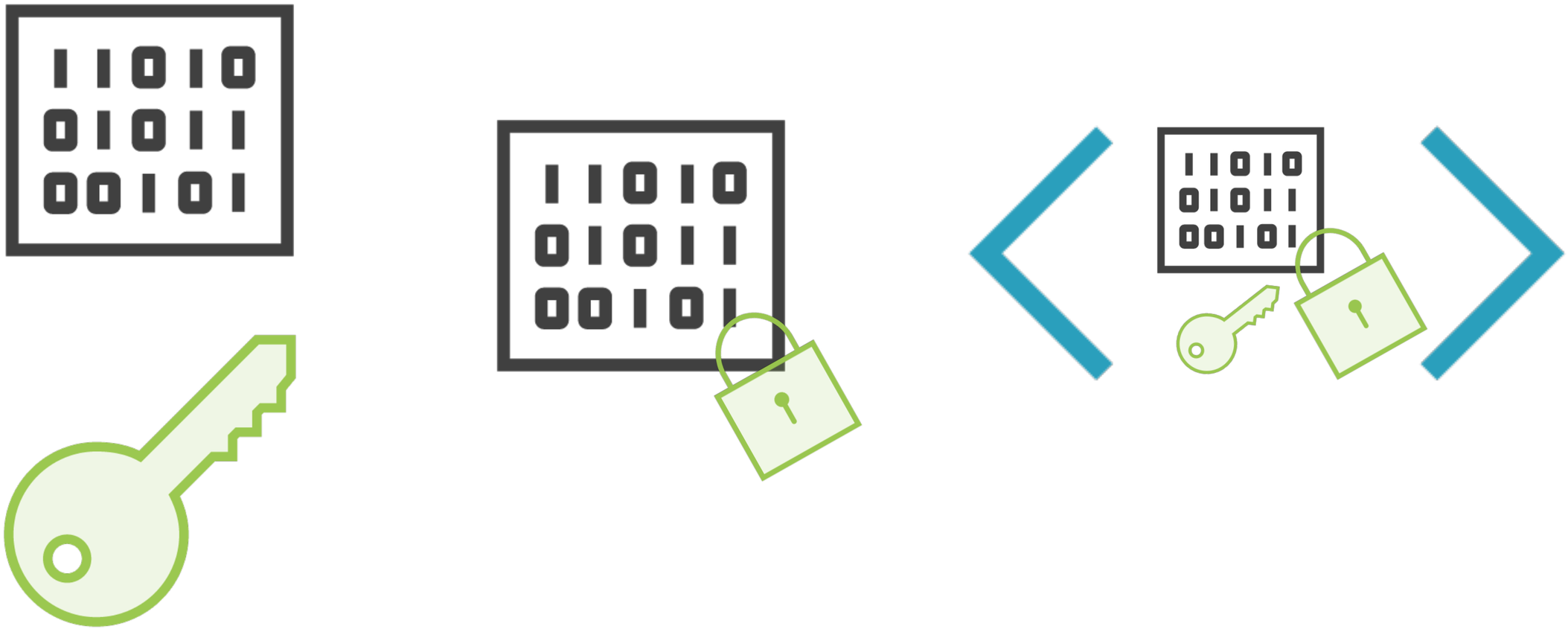


Unencrypted values in config file (Free!)



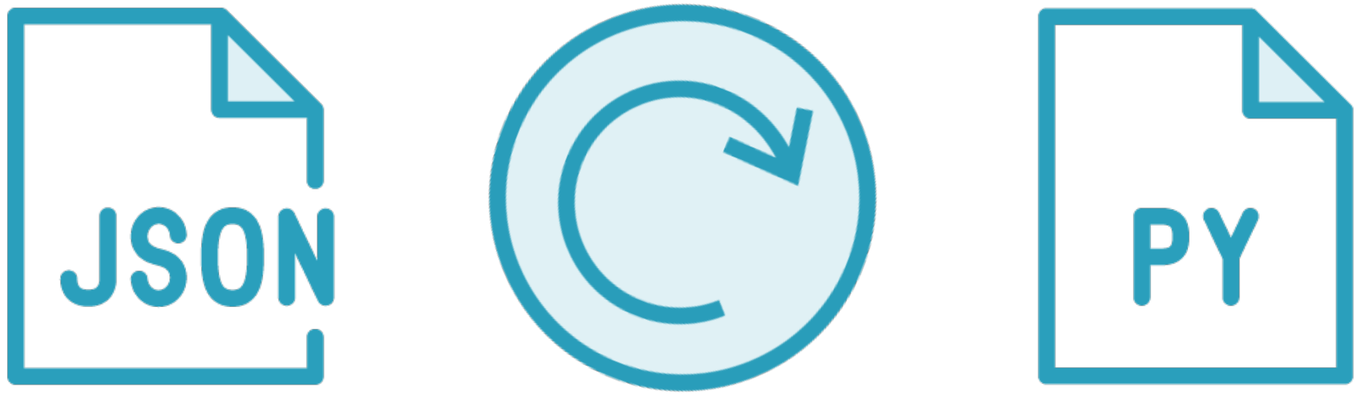
Encrypted values in config file (Cheap & more secure!)

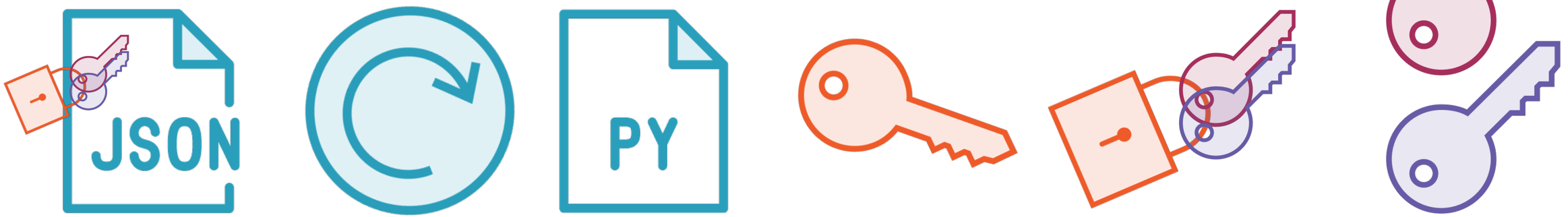
Data Encryption



AWS Key Management Service (KMS)

Config File Options





Demo

Preparing

- Get a Twitter account and API keys
- Setup Twython and Sparrow

Write Our Function

- Modify the Sparrow skeleton
- Test locally

Deploying Our Bot

- Creating a function package
- Setting event triggers

Demo

Preparing

- Gather credentials
- Download our dependencies

Write Our Function

- Modify the Sparrow skeleton
- Test locally

Deploying Our Bot

- Creating a function package
- Setting event triggers

Summary

What we've done

- Used an API with Lambda
- Deployed a function package
- Used libraries in our function

What's next?

- Lambda and other AWS services

Outline

Resources

- AWS Simple Email Service (SES)
- Skeleton code (Cuckoo)
- Boto 3 (AWS SDK for Python)
- Jinja2 (Templates)

Deployment

- Scheduled Event configuration - Cron

Amazon Simple Email Service (SES)

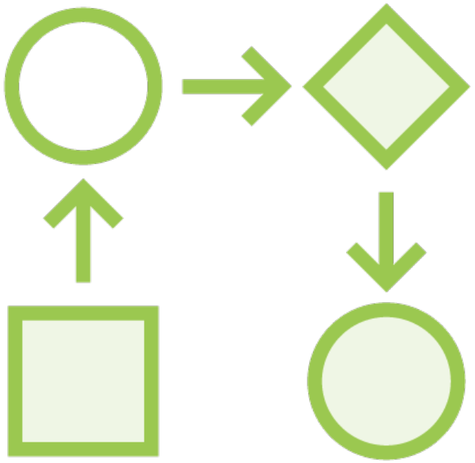
Inbound mail

Outbound mail

Benefits of SES

High deliverability

Sender statistics



Control flow

Monitoring

Cuckoo - Our Skeleton Code

Reminder types

Scheduled times

Email templates

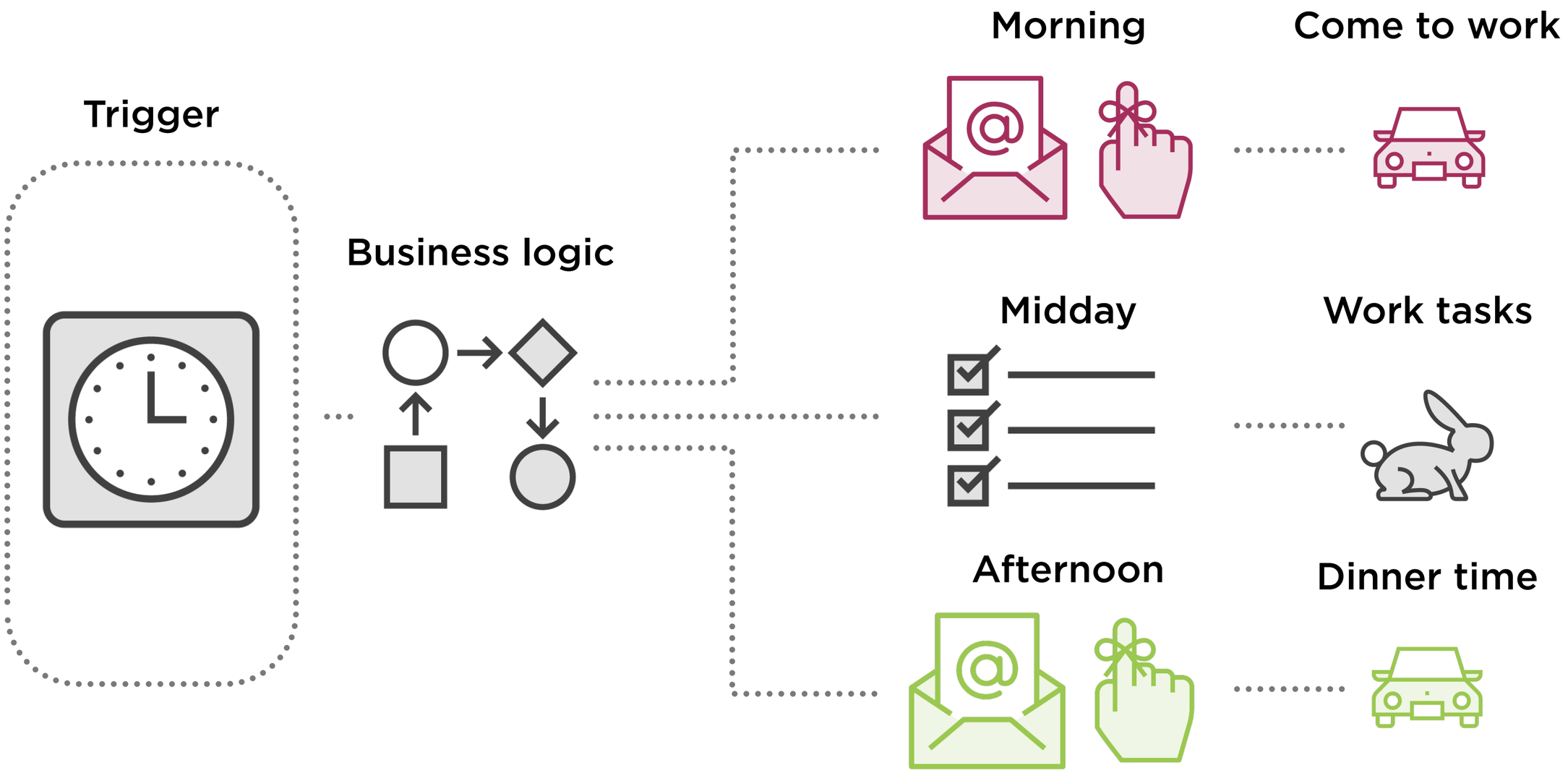
Mail groups

Other Resources

Python AWS SDK - Boto 3

HTML template - Jinja2

Workflow Automation



Demo

Preparing

- SES setup
- Download our dependencies

Write Our Function

- Modify the Sparrow skeleton
- Test locally

Deploying Our Bot

- Creating a function package
- Setting event triggers

Summary

AWS Services

- AWS service areas
- Simple Email Service (SES)

Resources

- Python Boto 3 SDK
- Jinja2
- Skeleton code

Deployment

- Event configuration