Challenges for Serverless Native Cloud Applications

Third International Workshop on Serverless Computing

Ben Kehoe
Cloud Robotics Research Scientist
AWS Community Hero
@ben11kehoe

2018-07-02
About me

• Cloud robotics: Connecting robots to the internet to help them do more and better things
• Cloud Robotics Research Scientist at iRobot (since Feb 2015)
• PhD in Mechanical Engineering from UC Berkeley, Dec 2014
  • “Brass”
Why serverless?
The reasons for serverless

What:
• Service-full + ephemeral compute
  • Not always F, not always aaS
• Resources billed → resources used
• Smaller, more abstract control plane

Why:
• Lower cost
• Lower operations burden
• Faster time to market
• Focus on business value
Serverless architecture

• Call graph → component graph

• Distributed system thinking
  • Traditionally occurs at system boundaries
  • Serverless: must be treated systematically

• Build robust-by-design systems
Before serverless
aka the dark ages

```python
def foo(input):
    quux = bar(input.baz)
    internalState.quux = quux

def bar(input):
    # do work
    return result
```
def handler(event, context):
    quux = Lambda.Invoke(
        'bar',
        event['baz'])
    DynamoDB.PutItem(
        'quux',
        quux)
Deployment
Deployment

• Red/black imposes requirements on clients
• Blue/green is the direction providers are headed

• Existing paradigm:
  • Blue/green controller is part of your component graph
  • Update component graph in-place
  • Controller manages roll-out
API

Function router

Function version

v1

v2
Function/code versions must be first-class citizens in infrastructure
Continuity of role may be necessary
Summing up

• Red/black is actually pretty easy for serverless, but harder on the service consumers

• Blue/green component graph gets complicated

• Existing deployment tools don’t represent intermediate states

• Native support for blue/green component graphs is critical
Questions?