AutoSpoting - an automated EC2 spot market bidder

GDG Berlin Golang - Clowdy Gophers

30 May 2016

Cristian Măgheruşan-Stanciu
HERE Maps, Berlin

About me

SysAdmin at HERE Maps, Berlin, supporting maps.here.com (https://maps.here.com)

- Background: networking, Linux, AWS, C/Shell/Perl/Ruby/Python
- Passionate about automation
- Spare-time gopher for a few months now, and I love it

Agenda

- The idea
- Implementation details
- Challenges and future plans
- Conclusions
- Q&A

The idea

AWS Spot Market Automation

Needed a non-trivial problem for learning Go in my spare time

- First discussed on an AWS Berlin meet-up
- Interesting, it got me thinking
- Decided to 'go build' it in my spare time

AutoSpoting

Simple solution to reliably use the AWS spot market, for production use-cases

- Easy to use
- Simple design and implementation
- Maximize cost savings without sacrificing high availability

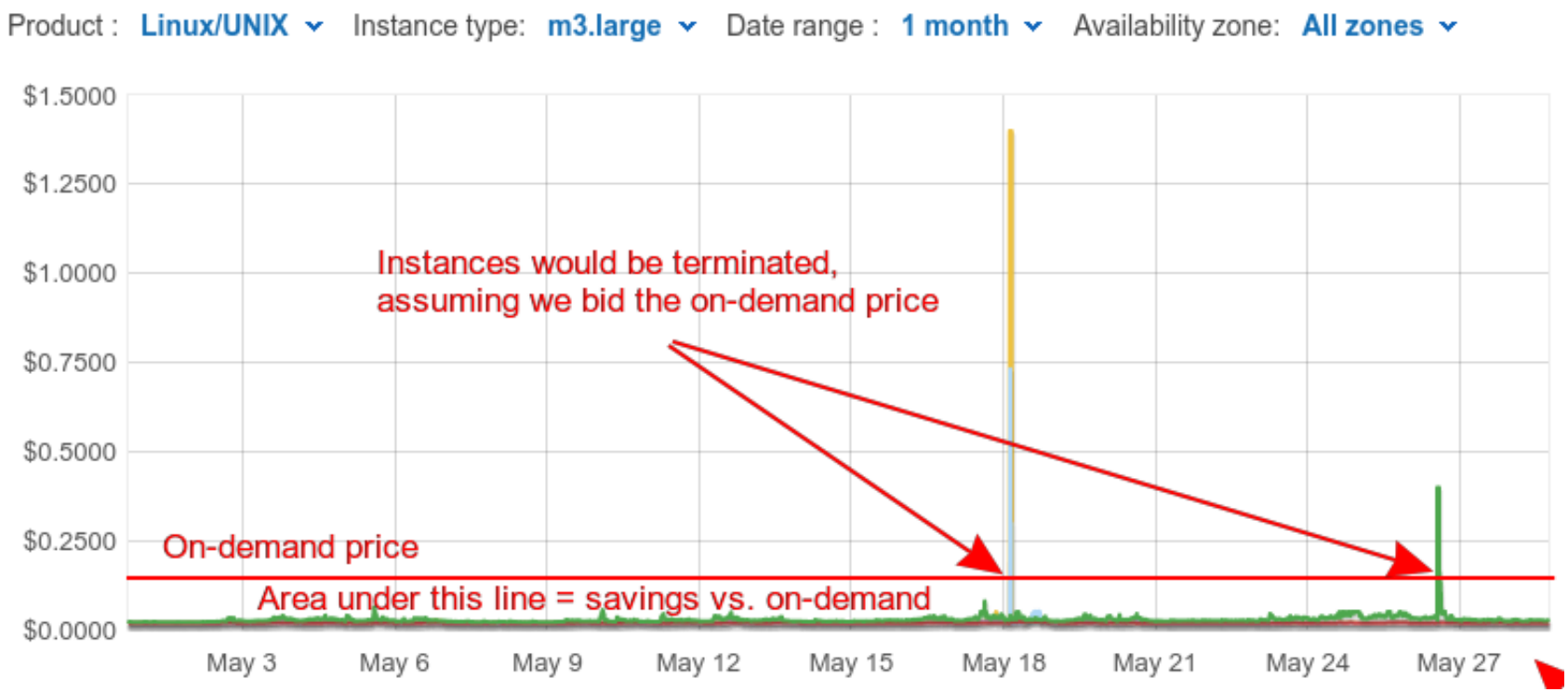
AWS Spot Market - Intro

Unused EC2 capacity sold to the highest paying bidders

- Prices based on supply/demand
- Considerable variation per region, instance type and even availability zone
- Not all of them are available on the market

AWS Spot Market - Typical Prices, Savings and Risks

- Starts from 10% the on-demand price, peaks as high as 10x
- Savings >80% of on-demand are common, or ~5x capacity for the same costs
- But **spot instances are terminated with a 2 minute notice when outbid!!!**



m3.large in Virginia, last 30 days

AWS Spot Market - Challenges

- Instances could be terminated at any time
- The application must be designed with this in mind
- Not applicable for everything
- Rewards cloud'y designs

AWS Spot Market - Achieving High Availability with Manual Bidding

- Place large bids and hope for the best
- Spread over multiple pricing zones
- Carefully pick region, instance type and availability zone
- AWS offers helper tools

Spot Bid Advisor

Region: US East (N. Virginia) OS: Linux/UNIX Bid Price: 25% On-Demand

Instance type filter:

vCPU (min): 4 Memory GiB (min): 4 Instance types supported by EMR

Instance Type	vCPU	Memory GiB	Savings over On-Demand*	Frequency of being outbid (month)	Frequency of being outbid (week)
m3.xlarge	4	15	85%	Low	Low
m1.xlarge	4	15	67%	Low	Low
i2.xlarge	4	30.5	89%	Low	Low
i2.8xlarge	32	244	88%	Medium	High
c4.xlarge	4	7.5	82%	Medium	Medium
c4.2xlarge	8	15	81%	Medium	Medium

AWS Spot Bid Advisor

AWS Spot Market - Automation Bidding Solutions

AWS offers just building blocks

- AutoScaling native integration - supports a single instance type
- SpotFleet - entirely different API, has some limitations (scaling, ELB...)

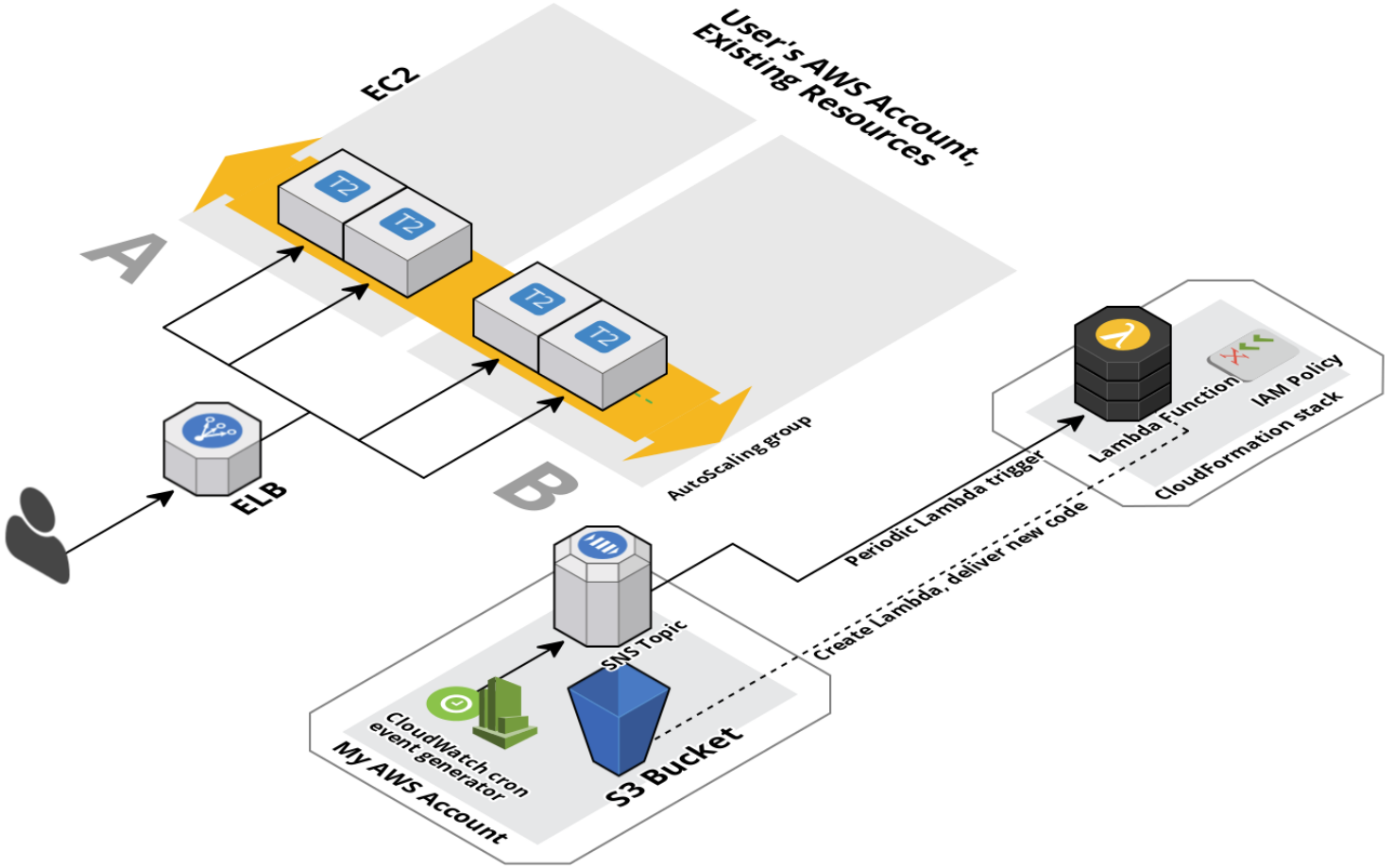
Also there are many 3rd party & custom tools

My Solution - AutoSpoting

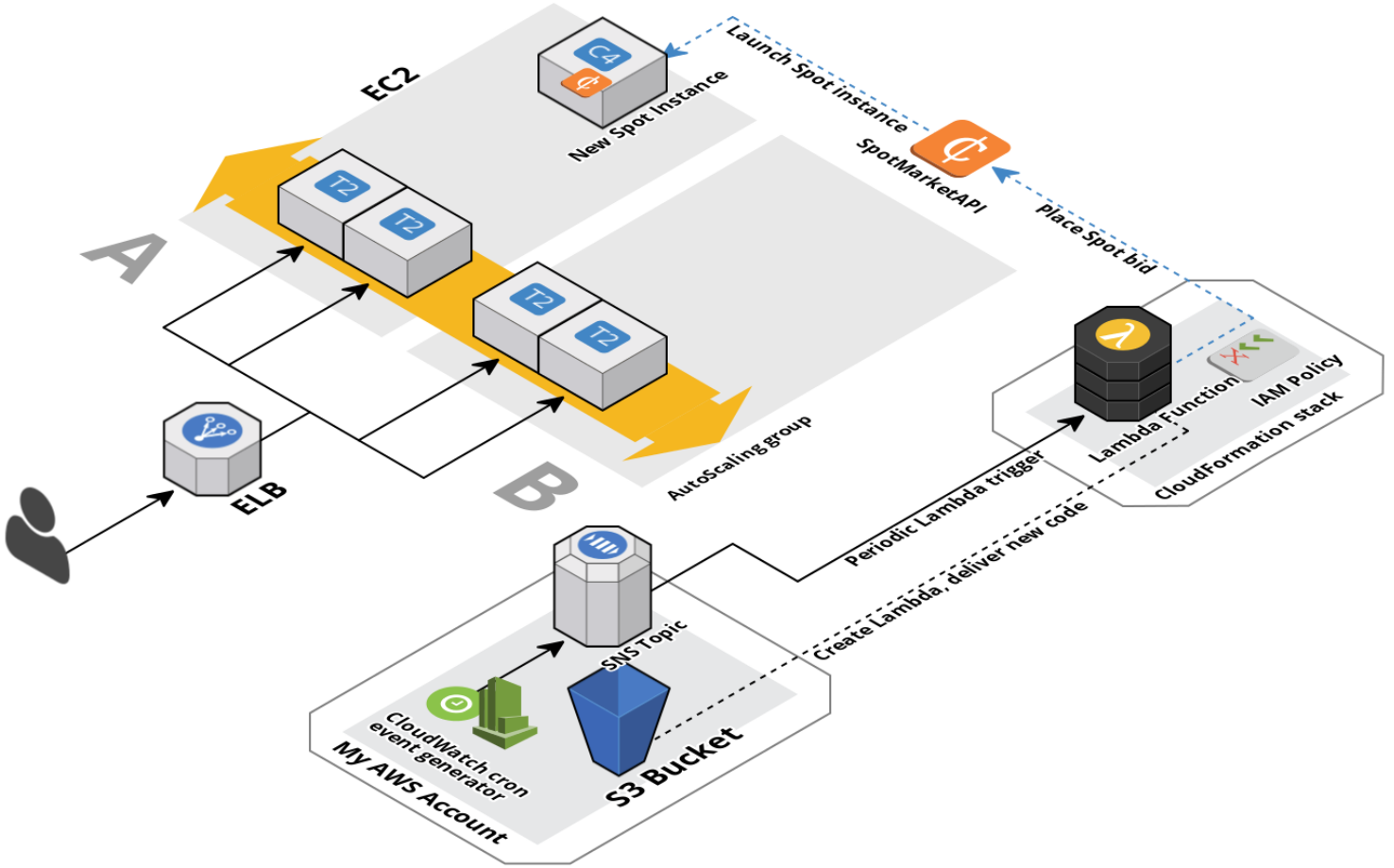
- Leverage recent autoscaling features to replace on-demand instances with spot equivalents
- Spread over multiple instance types in each availability zone
- AutoScaling handles instance terminations and ELB integration for the spot instances
- Easy to install and configure against existing AutoScaling groups
- Serverless and low overhead

Implementation details

Architecture

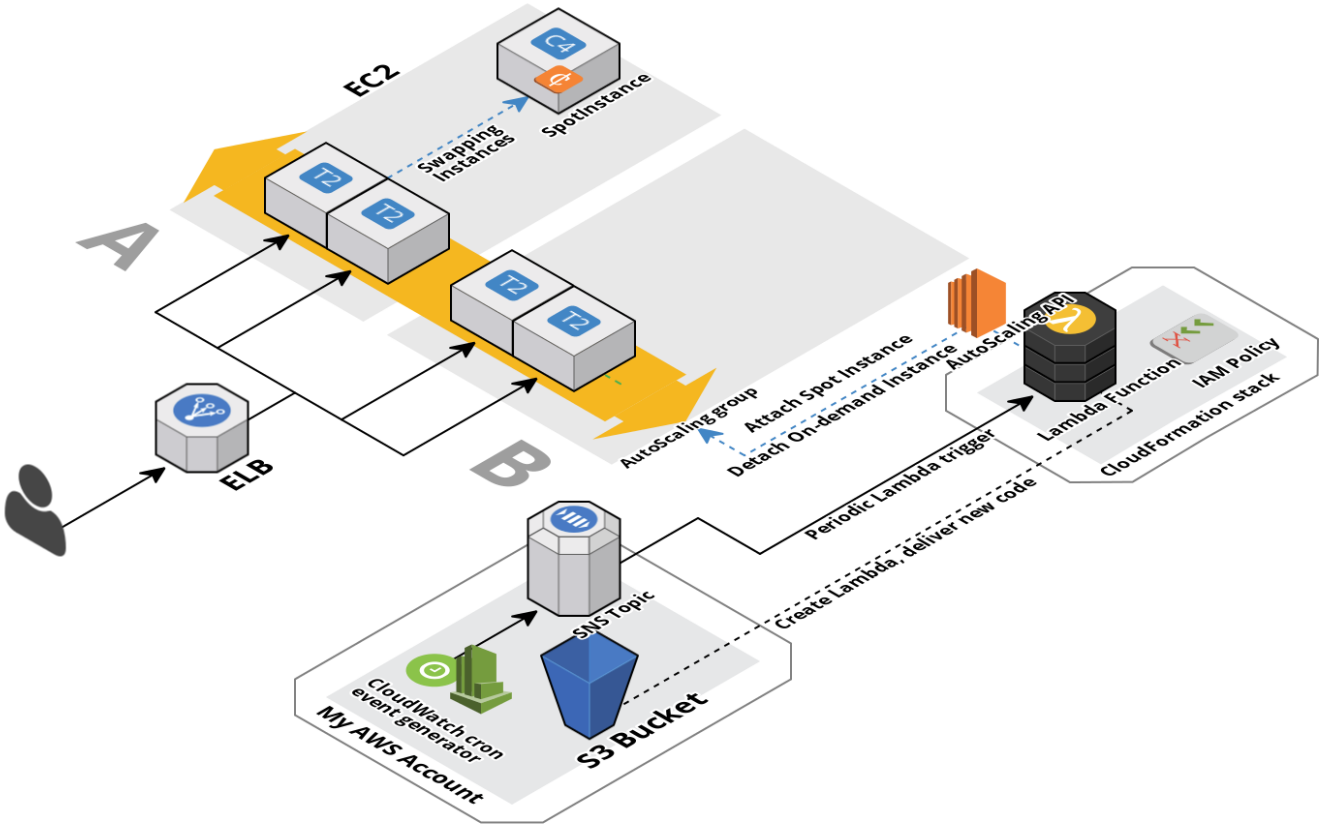


Workflow - Request New Spot Instance

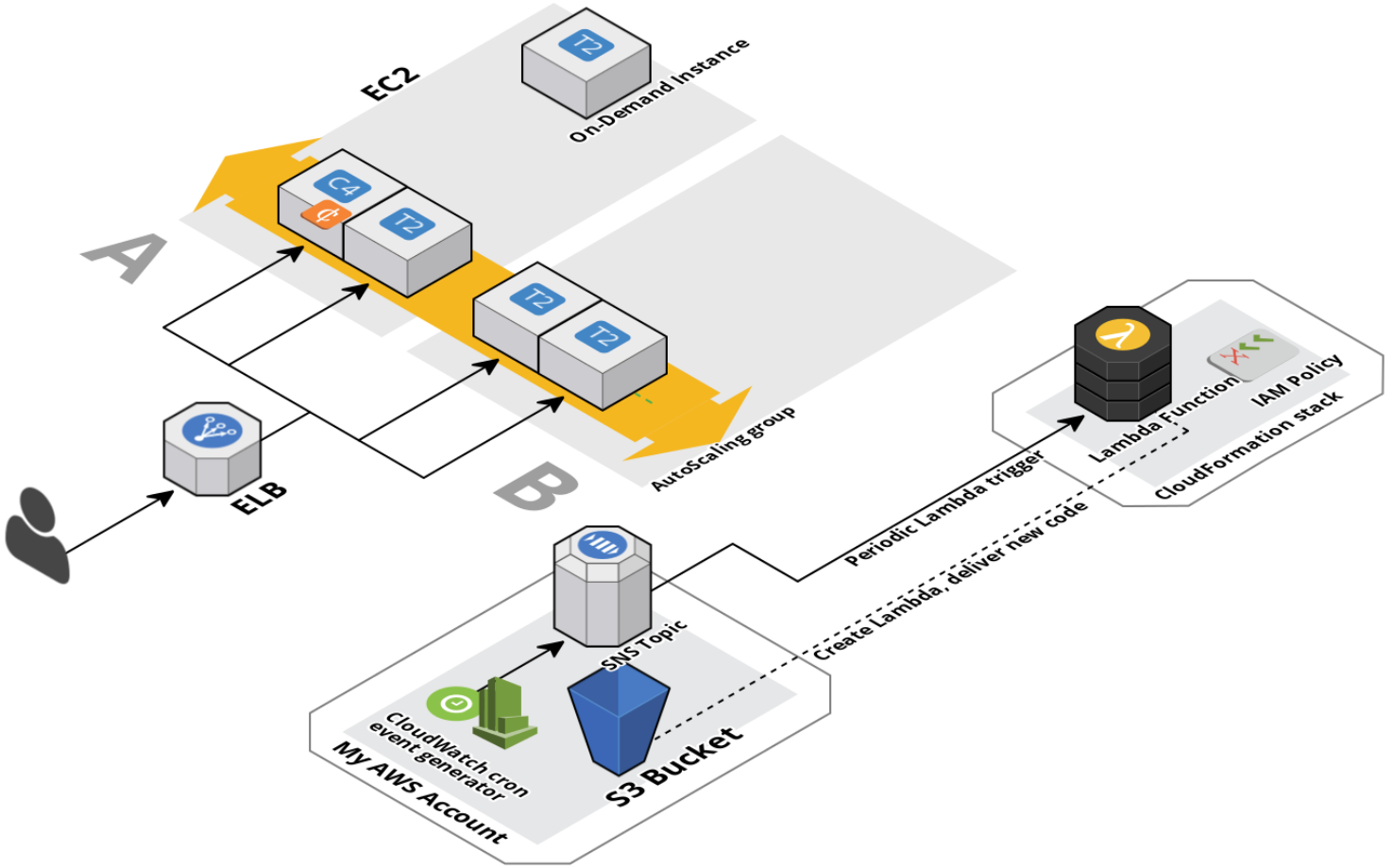


Workflow - Swap Instances

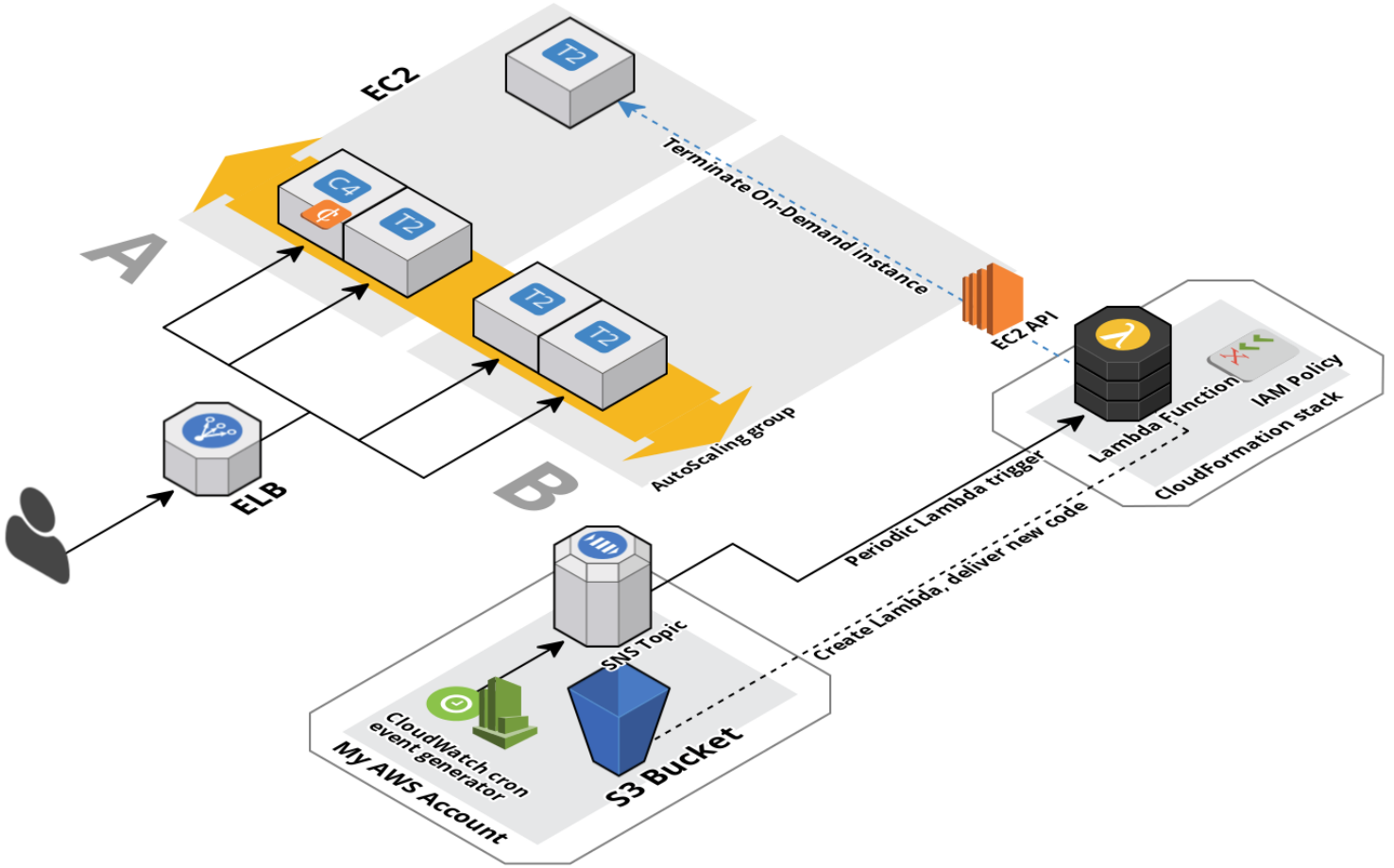
After the spot instance has been running for longer than the group's grace period



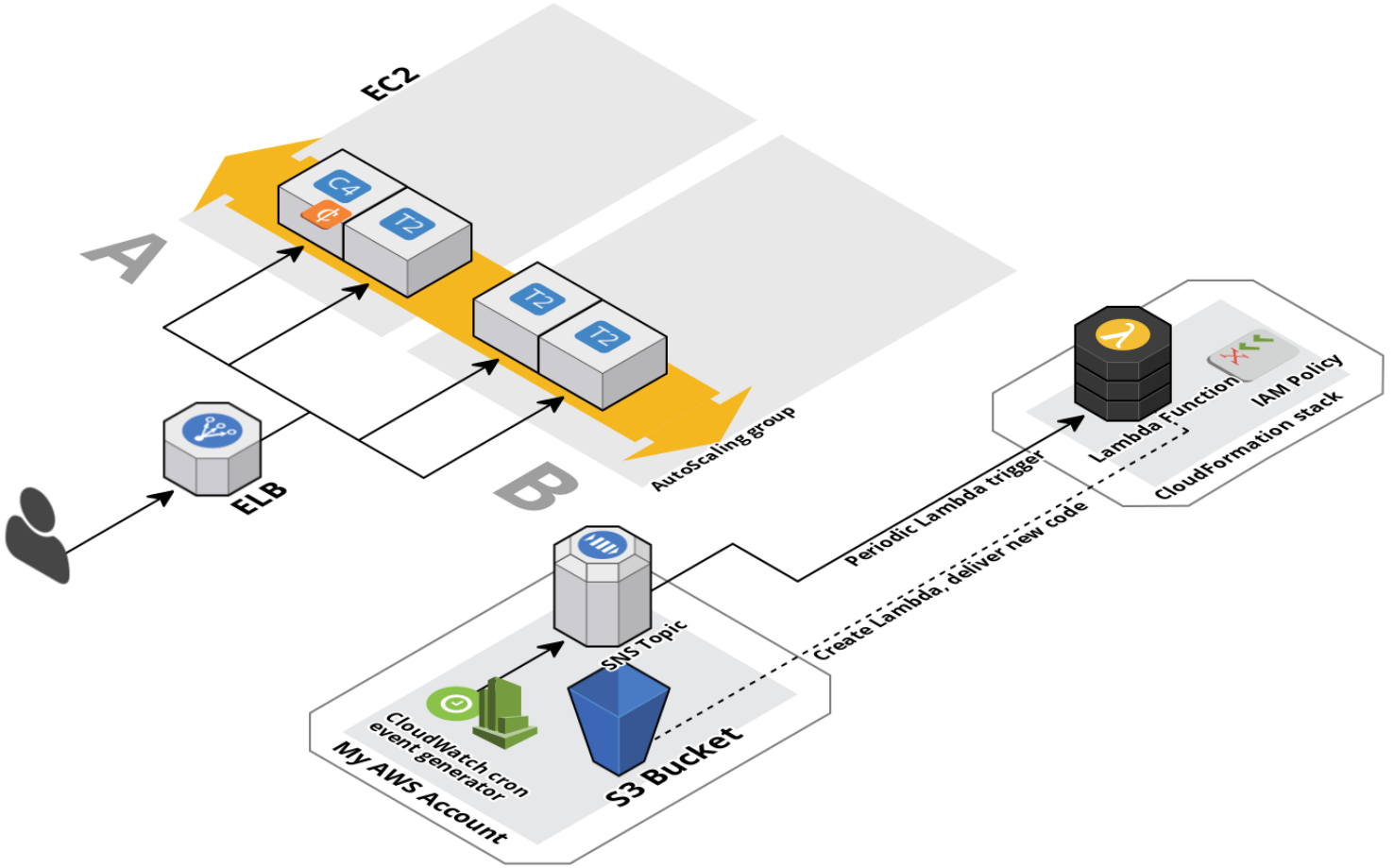
Workflow - Swapped Instances



Workflow - Terminate On-Demand Instance



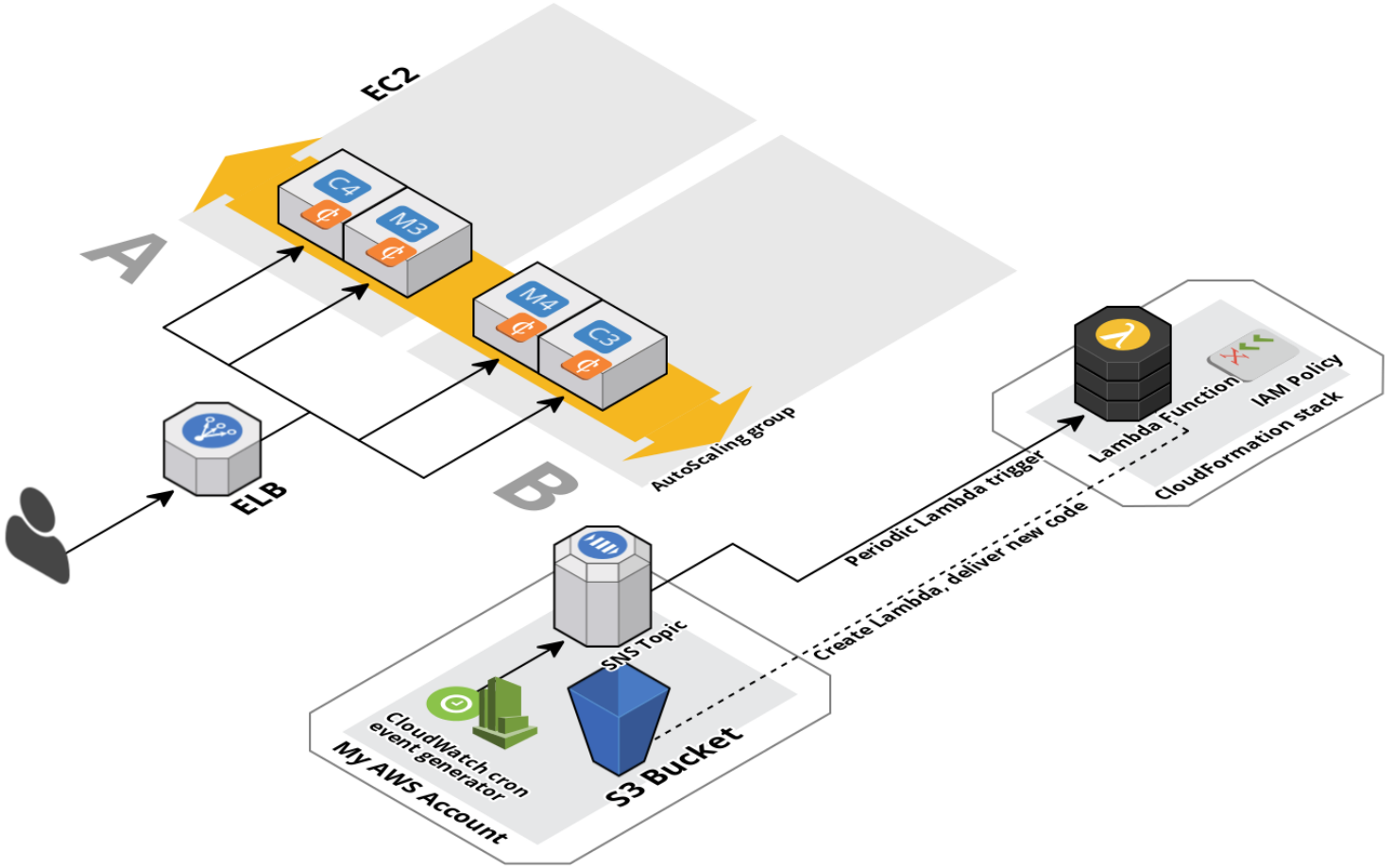
Workflow - Instance Replaced



Workflow - Loop until done

Repeat this until we have no on-demand instances left in the group

Workflow - Final State



Algorithm details

- The Lambda JavaScript shim automatically updates to the latest released binary that is implementing all the logic and runs it
- The Go program downloads data describing instance specs and on-demand pricing (compiled by ec2instances.info)
- Goroutines are then processing in parallel all the regions and enabled groups
- The spot instance type and bid price are determined for each group, based on the on-demand running instances and current spot prices
- Instance replacement is performed as described in the workflow

High Availability

Spot instance termination, when outbid

- Handled by AutoScaling like any instance failure
- AutoScaling will launch on-demand instances to compensate for lost capacity
- Those will in turn be replaced with spot instances later, as per the workflow
- Low likelihood of service downtime, with enough pricing zones and redundancy

Installation details

- Easy deployment via CloudFormation stack
- Enable/disable per AutoScaling group using a custom tag, off by default
- Have a look on my my blog (http://mcristi.wordpress.com) for detailed installation instructions

Challenges and future plans

Challenges

- Needs automated testing, currently I mostly do it manually
- Automated testing needs some internal redesign (WIP)
- Still trying to figure out testing with the AWS SDK for Go
- How to do table-structured tests with big/mocked data structures

Future Plans

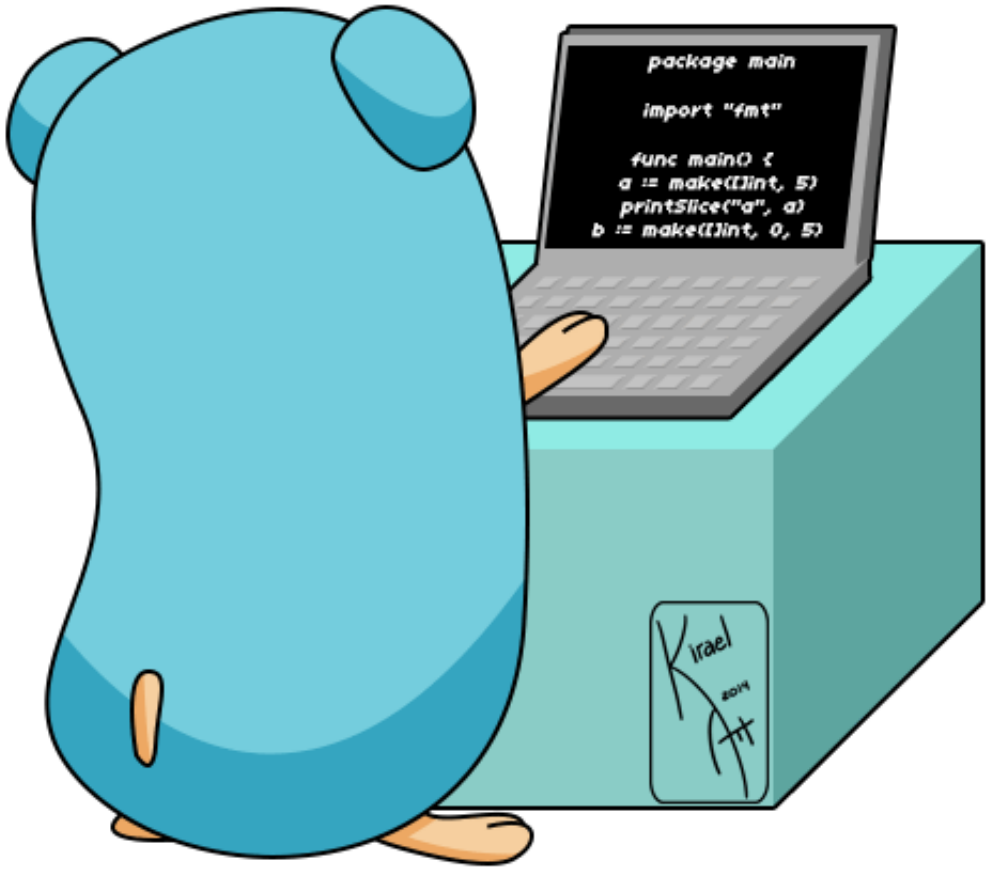
- Finish preparing for automated testing
- Write tests
- Use information about termination likelihood from the Spot Bid Advisor
- Make some parameters of the algorithm configurable
- React on the 2min termination notice, with code running on spot instances
- Allow keeping some on-demand capacity if configured
- Open source the code

Conclusions

What I like about Go

- Easy to learn
- Easy refactoring
- Easy to do concurrency
- Error handling
- Static binaries allow me to deploy easily

If you also have an interesting idea, 'go build' it!



Questions and Answers

Thank you

Cristian Măgherușan-Stanciu
HERE Maps, Berlin
cristian.magherusan-stanciu@here.com (mailto:cristian.magherusan-stanciu@here.com)
https://mcristi.wordpress.com (https://mcristi.wordpress.com)
@magheru_san (http://twitter.com/magheru_san)