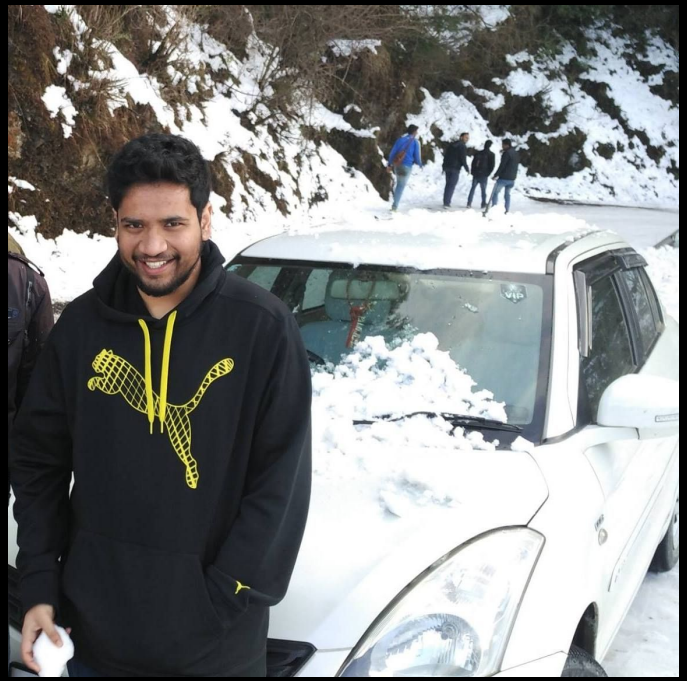
Understanding serverless architecture

—

Introduction



I am Jalem Raj Rohit.

Works on Devops and Machine Learning full-time.

Contributes to Julia, Python and Go’s libraries as volunteer work, along with moderating the Devops site of StackOverflow

What does serverless mean?

Serverless computing, also known as **function as a service** (**FaaS**), is a cloud computing code execution model in which the cloud provider fully manages starting and stopping of a function's container platform as a service (PaaS)

Setting the context

- Let’s assume our task here, is to move files from one S3 bucket to another, while changing the name of the files

Understanding “**function as a service**”

- Every serverless model has a function which is executed on the cloud
- These functions are executed depending on the activation of certain triggers [Display of triggers]

Understanding “manages starting and stopping of a function”

- The function is executed whenever one of it’s triggers are activated
- The function is stopped depending on the logic used inside it

Understanding “function's container”

- The functions are executed in containers
- This containers are shut down or thawed after the function execution is completed

Thus, “Look Ma, no servers”

- So, we are not running and maintaining any servers 24/7
- Everything, right from creating, provisioning of servers and execution of code is taken care in the cloud

```
> supply 2.
> Critical;Sun Aug 18 15:53:04 2013;Cannot communicate with power supply 2.
> Ok;System Boot;The chassis is closed while the power is off.
> Critical;System Boot;The chassis is open while the power is off.
> Critical;Sun Aug 18 15:53:07 2013;Power supply redundancy is lost.
> Critical;Sun Aug 18 15:53:07 2013;Power supply 2 failed.
> Ok;Sun Aug 18 15:53:12 2013;Power supply 2 is operating normally.
> Ok;Sun Aug 18 15:53:13 2013;The power supplies are redundant.
> Non-Critical;Wed Aug 21 23:10:12 2013;System level current is greater
> than the upper warning threshold.
> Critical;Wed Aug 21 23:10:16 2013;System level current is greater than
> the upper critical threshold.
> Non-Critical;Wed Aug 21 23:11:53 2013;System level current is greater
> than the upper warning threshold.
> Ok;Wed Aug 21 23:11:57 2013;System level current is within range.
> Non-Critical;Thu Aug 22 14:54:30 2013;System level current is greater
> than the upper warning threshold.
> Critical;Thu Aug 22 14:54:34 2013;System level current is greater than
> the upper critical threshold.
> Non-Critical;Thu Aug 22 14:55:05 2013;System level current is greater
> than the upper warning threshold.
> Ok;Thu Aug 22 14:55:09 2013;System level current is within range.
>
```

Advantages of serverless computing

- Less time maintaining servers, and more time cooking up awesomeness [Developer Productivity++]
- Lots of server cost saved for not running them ‘round the clock

Dis(Advantages) of serverless computing

- Functions are allowed to run for only a limited amount of time [Configs demo]
- No control over the container being spawned by the cloud provider [like the VPC, OS, etc]

Dis(Advantages) of serverless computing [contd.]

- Monitoring serverless services is very very very difficult
  - Especially, when they scale out to become **distributed, serverless services**
- Heavy workloads cannot be run [due to no control]

Lessons learned and pitfalls faced

- Next half of this talk would be about the lessons learned and pitfalls faced while building and scaling up serverless services

Expectations from the project

- Wanted to build a completely serverless end-to-end data pipeline
- Including extremely heavy computations like deep learning

Solving the “limited running time“ problem

- Each run of the pipeline would take atleast an hour to run
- So clearly, the 5 mins time limit is nowhere close to our expectations

Ansible to the rescue..

- Ansible is a tool which helps provision servers and run some tasks inside them
- So, created a server from the container
- Used it as Ansible's master for provisioning workers

Ansible to the rescue.. [contd...]

- Running Ansible in `nohup` mode in the master helped overcome the time limit
- Having Ansible kill all the servers after the pipeline executions made it completely serverless.

Solving the “no control on container” problem

- Security was the top priority for us, and there is no way to control the VPC of the container
- So, using Ansible to provision servers in specific subnets solved the problem

Horrors of distributed systems

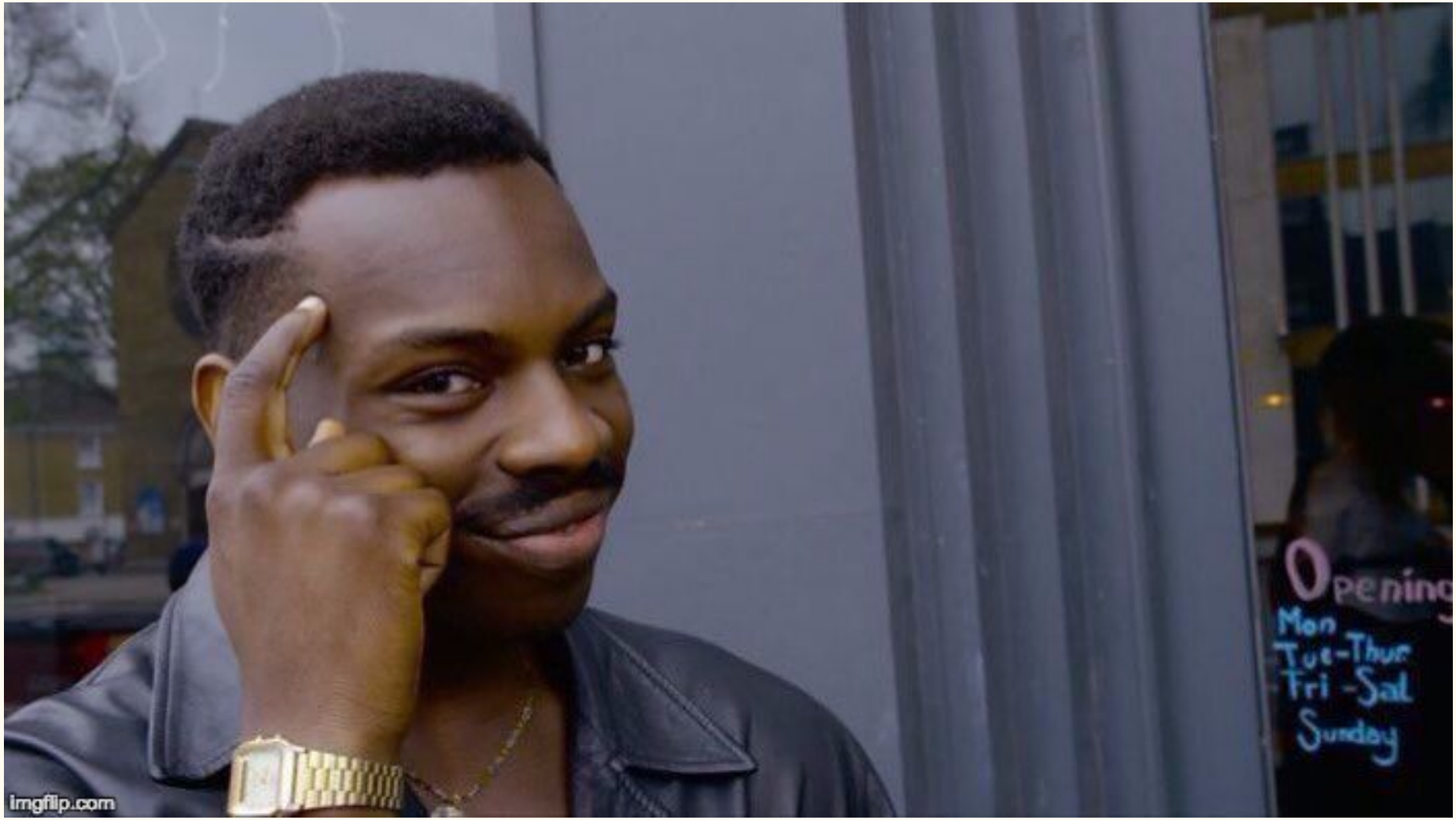
- Distributed systems is a very powerful paradigm, but they come with their own set of horrors
- What if a server(master/worker) goes down in between?
- What would happen to the data inside it?

Monitoring and logging is a monster now

- Monitoring a distributed, serverless system is an extremely difficult task
- Same applies for logging

But, but.... WHY?

- Where will the monitoring system lie? Would you have a server for that?
- A SERVER FOR MONITORING A SERVERLESS ARCHITECTURE?


Opening
Mon
Tue-Thur
Fri -Sat
Sunday
imgflip.com

What about Logging?

- Where would the logs be stored?
- Will each task send a log file? Or will the entire run be a single log file?

Answer

- Do most of the monitoring via the cloud providor's monitoring tool
- But, that tool might not have support for advanced monitoring

Answer [contd..]

- So, the horrors are usecase dependant.
- Zipping the logs from each worker after a complete run and sending to a db solved the purpose for us

Conclusions

- Serverless computing is awesome. Let’s do more of it

- However, it might not be the best choice for everyone. So, choose carefully.

Conclusions [contd ..]

- Scaling up serverless systems would involve the distributed systems paradigm, which is a fresh layer of hell
- Plan your monitoring very carefully