Why Serverless can work for enterprises?
I’m..

Satish Malireddi

- Principal Architect @ T-Mobile
- Part of Cloud Center of Excellence
- Serverless Advocate
- Focus: Cloud Technologies, Application Design & Architecture, Developer Experience & Tooling
Agenda

- T-Mobile’s Serverless Journey
- Challenges
- Business Case
- Toolset & Serverless Adoption
- Use Cases
- Adoption Plan (that worked for us!)
Serverless Journey...

2017
- Proof of concepts
- Microsites
- Policy & Compliance Manager
- **Jazz – Open source**

2018
- Thousands of Functions
- Millions of Invocations per day
- **Jazz - Production Ready!**

2019
- 3X growth in serverless workloads
- Tier-1 Applications
- Spike in $, #requests & #resources

2020
- Platform Strategy - Serverless First
- Billions of events per month
- Multiple apps in production
- **Jazz Adoption**
Our serverless adoption journey was not easy.

Because many developers think serverless...

- is new & immature
- has limitations
- requires a lot of architectural changes
- ecosystem is always changing
- might actually become expensive
Why should we really use serverless?
Business Case

Cost

Security

Agility
Business Case

$ Cost Control
- Reduce costs for suitable workloads
- Visibility into incurred costs

Security, Governance & Compliance
- Secure from day 1
- Complete visibility into what’s being built
- Implement guardrails through the platform

Reduce dependencies that hamper agility
- 100% Automation
- Ease of Use
- Improved Developer Experience
- Training
What Developers want?

- Agility
- Faster Time to Market
- Ease of Use

What Management wants?

- Governance
- Visibility
- Compliance
- Standardization
- Guardrails
- Process Control
We built Jazz, a Serverless Development Platform that enables developers to build secure, compliant serverless apps that are operationally ready from day one!

https://github.com/tmobile/jazz
Breaking it down

- Accelerate Serverless Adoption
- Built around two themes –
  - Ease of use
  - Build compliant applications in the cloud
- Enterprise processes are 100% automated
- Self-Service enabled to reduce dependencies
- Bridge gaps between actual serverless promise and the reality
- Keep developers & management happy
Features

- CI/CD
- Standards & Security Controls baked in
- Multi Tenancy
- 1-Click Environments
- Best practices through code templates (application marketplace)
- Governance & Compliance
- Log collection, aggregation & analytics
- Monitoring – Metrics, Dashboards & Alerts
- Enterprise Integrations through extensions
- Abstract Complexity with Cloud Provider solutions
How did Jazz help with adoption

- Improved time to market
- Faster access to the cloud
- Lower environment creation is as easy a simple "git commit"
- Best practices are being shared
- Developers are talking to each other
Top 5 Usecases

- Single purpose APIs
- Static Websites
- Event driven applications/functions
- Scheduled functions
- Data transfer/manipulation/processing jobs
Use cases

Config Management Service

List-Based Facts Service

Facts-Lang Service
Pacbot: Policy as Code bot

Use cases
**Use cases**

**Jazz - Event Flow**

**Producers**
- Jazz APIs
- Jenkins
- Bitbucket/SCM
- Other Integrations

**Consumers**
- /jazz/services
- /jazz/deployments
- /jazz/assets
- /jazz/environments
- Events DB

**Amazon EventBridge**
- service_events
- deployment_events
- asset_events
- environment_events
- all_events

**Benefits:**
- Less Code
- Less processing time
- Better visibility
- Less cost
- Scalable
- Easier to add more consumers
- Easier to add new types of consumers (Step Functions)

**Process & Re-Publish?**
- Read Failed Events
- DLQ (one or more)
- Failed Events

**Dashboard**
Challenges still remain..

- Difficult to change developer mindset
- Looking for Lift-n-Shift
- Constrained by lot of factors when making modern design choices
- Developers are not aware of Op-Ex savings
- Lack of training, not being up-to-date with technology
- Technical limitations with cloud offerings (might go away with time)
Plan that worked for us

- Identify **use cases** that are best suited and go after them
- Don't over engineer! Serverless might not be the **perfect fit** for all your applications
- Provide **visibility into cost** savings/estimates during the development phase
- **Training** (most of the time its about people not being aware) & make it a continuous exercise
- Have **CoE teams**: Dedicated folks who can experiment, learn, train others, identify tools to empower developers
- Create framework for developers so that they can **experiment easily** within controlled guard rails
- If you are developing abstractions, **listen** to your developers to solve their pain points and improve developer experience