Cloud Native Applications

主讲人：Capital One 首席工程师 Kevin Hoffman

Agenda

- Define Cloud Native
- Cloud Native practices and the “15 factors”
- From Monoliths to the Cloud
- Q&A

What is Cloud Native?

- Contract between application and infrastructure
- Designed to scale elastically and horizontally
- No host dependencies
- Application conforms to the “12 (or 15) factors”
- Container-based*

Team Discipline

云道

- Structure around rapid, continuous delivery
- Embrace agility
- Embrace the cloud
- Adopt practices aimed at thriving on cloud platforms

The Way of the Cloud

云道

- Favor Simplicity
- Test First, Test Everything
- Release Early, Release Often
- Automate Everything
- Build Ecosystems, not Monoliths

15 Factors of Cloud Native 十五个因素

- Based on Heroku’s original “12 factor” list (12factor.net)
- Characteristics of a cloud native application
- Some easier to adopt than others
- Adopting all factors pays huge dividends

One codebase, One App

第一

- Single version-controlled codebase, *many* deploys
- Multiple apps should not share code
  - Microservices need separate release schedules
  - Upgrade, deploy one without impacting others
- Tie build and deploy pipelines to single codebase

API First

第二

- Service ecosystem requires a contract
  - Public API
- Multiple teams on different schedules
  - Code to contract/API, not code dependencies
- Use well-documented contract standards
  - Protobuf IDL, Swagger, Apiary, etc
- API First != *REST* First
  - RPC can be more appropriate in some situations

Dependency Management

第三

- Explicitly declare dependencies
- Include all dependencies with app release
- Create immutable build artifact (e.g. docker image)
- Rely on smallest docker image
  - Base on scratch if possible
- App cannot rely on host for system tools or libraries

Design, Build, Release, Run

第四

- Design part of iterative cycle
  - Agile doesn’t mean random or undesigned
- Mature CI/CD pipeline and teams
  - Design to production in *days* not months
- Build creates immutable artifact
- Release *automatically* deploys to environment
  - Environment contains config, *not* release artifact

Configuration, Credentials, Code

第五

- “3 Cs” volatile substances that explode when combined
- Password in a config file is as bad as password in code
- App must accept “3 Cs” from *environment* and only use harmless defaults
- Test - could you expose code on github and not reveal passwords, URLs, credentials?

Logs

第六

- Emit formatted logs to stdout
- Code should not know about destination or purpose of log emissions
- Use downstream log aggregator
  - collect, store, process, expose logs
  - ELK, Splunk, Sumo, etc
- Use *structured logs* to allow query and analysis
  - JSON, csv, K=V, etc
- Logs *are not* metrics

Disposability

第七

- App must start as quickly as possible
- App must stop quickly and gracefully
- Processes start and stop all the time in the cloud
- Every scale up/down disposes of processes
  - Slow dispose == slow scale
- Slow dispose or startup can cause availability gaps

Backing Services

第八

- Assume all resources supplied by *backing services*
- Cannot assume mutable file system
  - “Disk as a Service” (e.g. S3, virtual mounts, etc)
- Every backing service is *bound resource*
  - URL, credentials, etc -> environment config
- Host does not satisfy NFRs
  - Backing services and cloud infrastructure

Environment Parity

第九

- “Works on my machine”
  - Cloud-native anti-pattern. Must *work everywhere**
- Every commit is candidate for deployment
- Automated acceptance tests
  - Provide no confidence if environments don’t match

Administrative Processes

第十

- Database migrations
- Run-once scripts or jobs
- *Avoid* using for batch operations, consider instead:
  - Event sourcing
  - Schedulers
  - Triggers from queues, etc
  - Lambdas/functions

Port Binding

第十一

- In cloud, infrastructure determines port
- App must accept port assigned by platform
- Containers have internal/external ports
  - App design must embrace this
- Never use reserved ports
- Beware of container “host mode” networking

Stateless Processes

第十二

- What is stateless?
- Long-term state handled by a backing service
- In-memory state lives *only* as long as request
- Requests from same client routed to different instances
  - “Sticky sessions” cloud native anti-pattern

Concurrency

第十三

- Scale horizontally using the process model
- Build disposable, stateless, share-nothing processes
- Avoid adding CPU/RAM to increase scale/throughput
- Where possible, let platform/libraries do threading
  - Many single-threaded svcs > 1 multi-threaded monolith

Telemetry

第十四

- Monitor apps in the cloud like satellite in orbit
- No tether, no live debugger
- Application Perf Monitoring (APM)
- Domain Telemetry
- Health and system logs

Authentication & Authorization

第十五

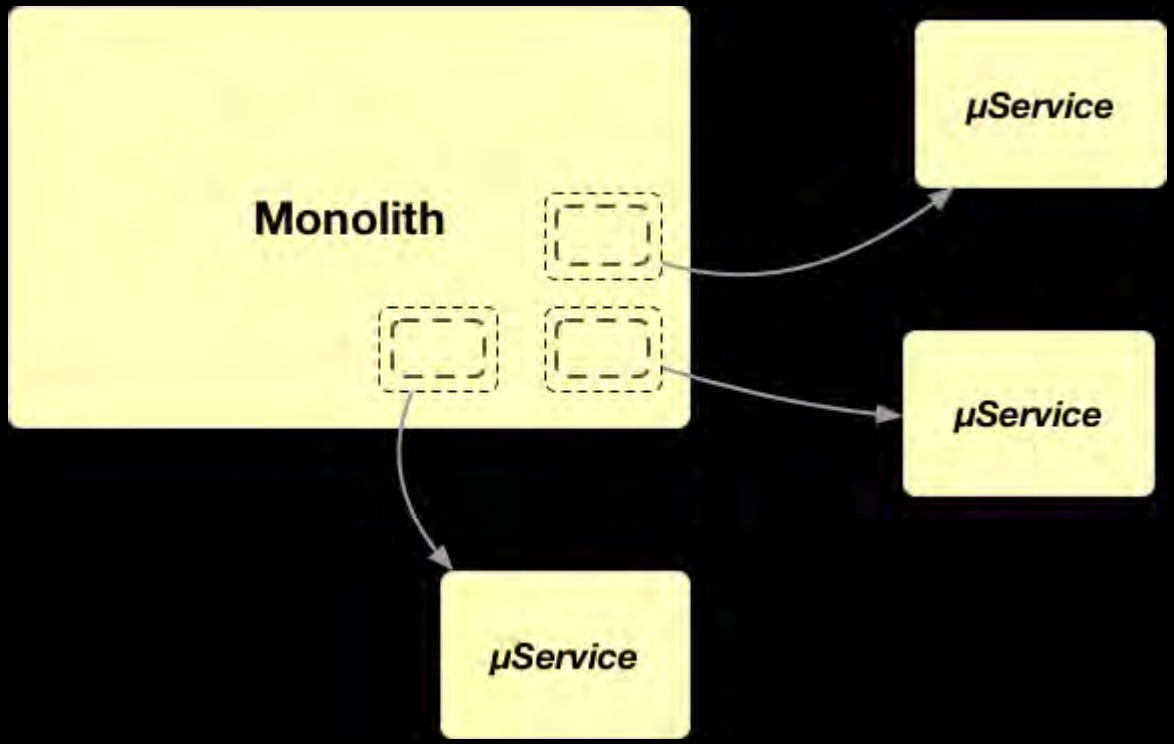
- Security should never be an afterthought
- Auth should be explicit, documented decision
  - Even if anonymous access is allowed
  - Don’t allow anonymous access 😄
- Bearer tokens/OAuth/OIDC best practices
- Audit all attempts to access

Migrating Monoliths to the Cloud

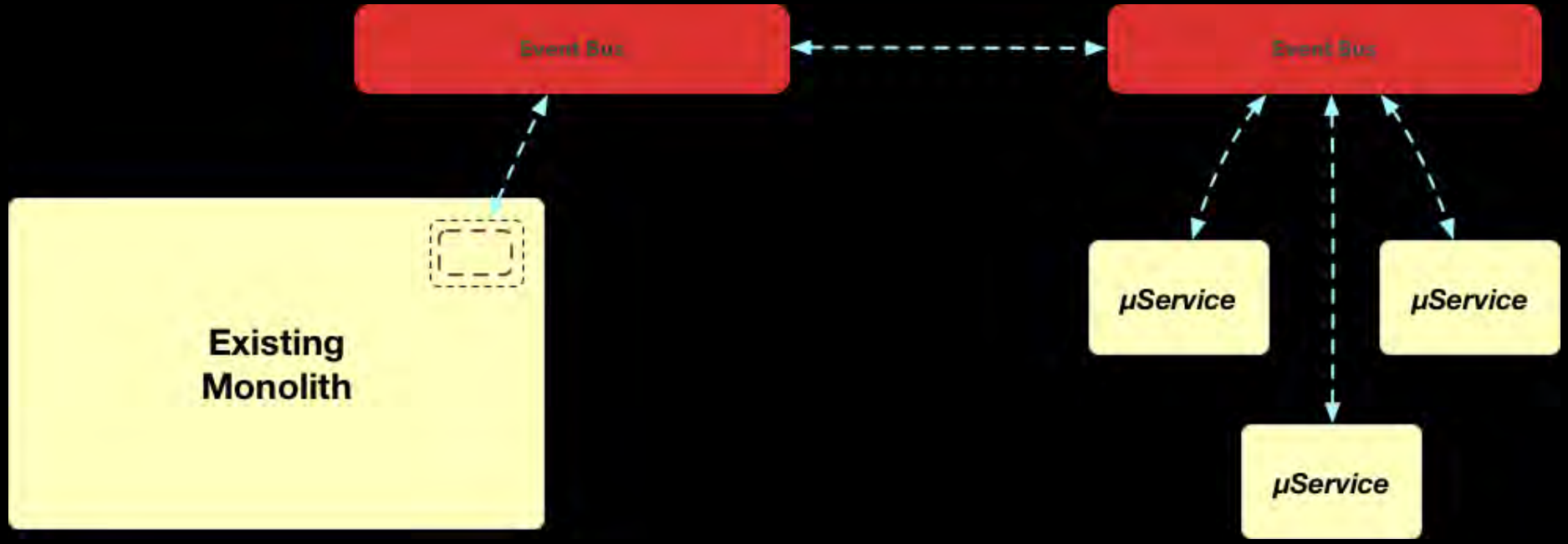
- Make a rule - *stop adding to the monolith*
  - All new code must be cloud native
- Prioritize features
  - Where will you get most benefit from cloud native?
- Come up with a plan
  - Decompose monolith over time
  - Fast, agile iterations toward ultimate goal
- Use multiple strategies and patterns

Monolith Migration - “Lift and Shift”

Monolith Migration - Isolate and Break



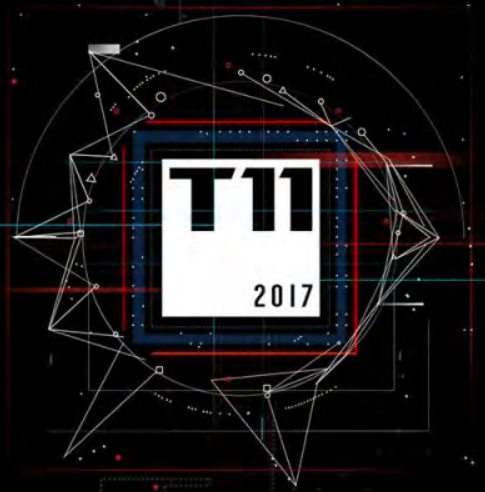
Monolith Migration - Event Sourcing



Cloud Native Go

- Lightweight
- Easily learning curve
- Compiles to native binaries
- Very fast
- Large, thriving, engaged community
  - http://gopherize.me

Questions 问题



THANKS

TalkingData