Electric Cloud

Microservices and Docker at Scale

The PB&J of Modern Application Delivery

Avan Mathur
Product Manager, Electric Cloud
@avantika_ec

What Am I Talking About?

- Microservices: what's cool about them
- Containers: what's cool about them
- Why are they often mentioned in the same sentence?
- Challenges to good outcomes
- Challenges at enterprise scale
- Best practices to improve outcomes

What are Microservices?

- A pattern for building distributed systems:
  - A suite of services, each running in its own process, each exposing an API
  - Independently developed
  - Independently deployable
  - Each service is focused on doing one thing well

"Gather together those things that change for the same reason, and separate those things that change for different reasons."

– Robert Martin

Things I’m Not Trying To Say/Imply/Spend Time On

- Microservices should be used by everyone, for everything
- If it’s not containerize-able, it’s crap
- How to solve all the tricky microservices problems in just one line of Haskell
- “My Kubernetes cluster beat up your Docker DataCenter and took its lunch money and gave it to Mesos”

Why Microservices?

- Divide and conquer complex distributed applications
- Loose coupling; each service can:
  - Use the tooling that's appropriate for the problem
  - Scale independent of other services
  - Have its own lifecycle independent of other services
- Easier to scale teams and scale applications
- Smaller more autonomous teams are more productive
- Better resource utilization

Why Containers?

- Containers provide a well-defined, isolated runtime environment
  - *-as-code: Dockerfile, docker images, repository
  - Isolated: toolchain & dependencies are in the container
  - Can be immutable
- Environment consistency: ship the environment instead of “artifact + giant readme”
- Move the container through the pipeline instead of the artifact
- Per-container overhead << per-VM overhead
  - Scale up & down much easier/faster – seconds instead of minutes
  - Better utilization
- Container hosting services provide cool/necessary features
  - Auto scaling, monitoring, resiliance
  - Some orchestration

Why Microservices in Containers?

- 2002: One service per metal box
  - “I remember my first dual-core box, too!”
  - “Why is that 32-core server idle all the time? Can I have it?”
- 2007: Hypervisor + 1 VM + Multiple services in that VM
  - “Yeah, can’t run ServiceA and ServiceB side by side, conflicting versions of…”
  - “Yeah, we did that until ServiceC filled up /tmp and took down ServiceD"
  - “Yeah, we tend to run ServiceE by itself once we’re past QA”
- 2012: Hypervisor + Multiple VMs + 1 Service in each VM
  - “Yeah, each VM OS has a copy of that in memory, so…”
- 2013: Containers: run multiple services in isolation without the OS overhead

The PB&J Part

- Microservice: one process that solves a single problem
- Container: a process and its associated dependencies with minimum overhead
- Decent overlap between common challenges and best practices

Challenges

Challenges Common to Both

- Complex tools for solving complex problems
- It isn't necessarily *easier* to do it this way…
- Orchestration of the software pipeline – more moving parts
- Increased testing complexity
- Increased deployment complexity
- Monitoring, logging, remediation are critical and more difficult
- Diverse toolchains, architectures, and environments
- How do you deploy updates to an application that has microservices running in containers as well as legacy components?

Microservices Can Be Challenging

- Distributed Systems Are Hard
  - Service composition is tricky to get right, can be expensive to change
  - Inter-process failure modes have to be accounted for
  - Abstractions look good on paper but beware of bottlenecks
- State management – transactions, caching, and other fun things
- *Team-per-service* or *Silo-per-service*? + Conway's Law
- Legacy apps: Rewrite? Ignore? Hybrid?
- Good system comprehension is key
- Your service might be small, but how large is its deployment footprint?

Microservice Challenges (continued)

- “Every outage is like a murder mystery!”
- “But my service relies on version X of ServiceA and now I’m down”
- “Wait, how many processes do I need to start for this test?”
- “So then requestId 0xf00dfee8 in the log on ms-app-642-prod becomes messageId 1125f34c-e34e-11e2-a70f-5c260a4fa0c9 on ms-route-669-prod?”
- Network issues – bandwidth, latency, reliability

Containers Can Be Challenging

- State/volume/data management
  - “Wait, I can’t even write a log file inside the container file system?”
  - “The auditors asked for logs, but we removed those containers”
- Security
  - What do you mean it runs as root?
  - Containers are black boxes to opsec: less control, less visibility inside the container, existing tools may not be container-savvy
  - How do you secure every random image that comes out of dev?
- Monitoring: what does it mean when containers come and go all the time?
- What hosting service should I use?
- Ecosystem is moving very fast
- Sprawl: it’s not just for VMs anymore

Challenges at Enterprise Scale

- Technology maturity
- Forrester 2015 Container Challenges
  - Security, Variable performance, Integration with existing tools & processes
- Large Fortune 20 Bank: 6,000 applications
  - Motivated by container efficiencies, ease of scaling, ease of replicating
  - Maybe only 20% of apps are appropriate for containerization
  - Not rewriting everything, but new development follows container-friendly approaches
  - “At least one of everything” can complicate life
- Governance/audit can get more complicated
- Bi-modal/multi-modal software delivery: monoliths, micro-services, and everything in between, for the foreseeable future

Best Practices

Common Best Practices

- Your Automated Software Pipeline Is Your Friend™
  - Ideally, one platform handles all your software delivery
  - How's your test coverage?
  - Are your tests automated? Really automated?
- Self-service automation/ChatOps approaches
  - Reduce onboarding time, waiting, complexity
- Service discovery
- Consistent logging & monitoring output across services/containers
  - Does your monitoring/perf tool help you track a request through the system even as it bounces between different services?

Microservices Best Practices

- Develop domain knowledge
- Avoid premature decomposition
  - If starting from scratch, keep it modular and split things up as your understanding of the problem domain evolves
- Be rigorous in handling failures (consider using, e.g. Hystrix to bake in better resiliency)
- Be flexible about staffing and organization

Microservices Best Practices (continued)

- Independent delivery pipeline for each service
  - Test independent of other services
  - Static analysis, security validation, perf testing
  - Integration pipelines further downstream

Container Best Practices

- Use an enterprise container registry
- Container-native monitoring
- State: what kinds and where and how to persist it?
  - Business data, reference data, compliance data, logs, diagnostics
  - There are open source and commercial offerings to help
- Security
  - Sign & scan images, validate libraries, etc.
  - Harden the container environment as well
  - Drop privileges early, use fine-grained access control so it’s not all root
  - Your Automated Software Pipeline Is Your Friend™

Container Best Practices (continued)

- What's running where and why?
- Understand platform dependencies that might limit you
- Avoid image bloat
- Immutable images?
- Do you really need an SSH daemon in that image?

https://neo4j.com/blog/managing-microservices-neo4j/

Electric Cloud

Thank you!

Questions?