Cloud Oriented Programming

Vishwas Lele
@vlele,
Applied Information Sciences

Cloud Oriented Programmming

Vishwas Lele
@vlele
Applied Information Sciences

Level: Intermediate, etc.

Cloud is tomorrow’s IT backbone

Key Tenets of Cloud Computing

- On Demand / Automated
- Utility Billing / Pooled Resources
- Commodity / Homogenous hardware
- Scale-out over Scale-up
- Proliferation of Services / Cadence of release

But do these tenets impact
me as a developer?

Well, Tenets Only Go so Far…

- On Demand / Automated
  - Think of infrastructure as code
- Utility Billing / Pooled Resources
  - But please be cognizant of the costs
- Commodity hardware
  - But prepare to handle failures
- Scale-out over scale-up
  - Think of Cloud as the OS
- Proliferation of Services / Cadence of release
  - More APIs learn ☺

Exponential Back Off

Back off, cool down and try again

Retry Logic

- Often necessary to retry an operation
- Especially in the cloud due to transient faults or other latencies
- Defensive programming
- Pay attention to SLAs
- Test your retry logic
- Think about overall resilience

Demo

Progressive Backoff

Reimagine the Exception Handler

What we anticipate seldom occurs; what we least expected generally happens

- Benjamin Disraeli

Exception Handling

- The basics remain the same in the cloud
- Follow well-established guidance for exception handling
  - Exception Hierarchy
  - Fail fast
  - Consider the performance impact
  - Tester-Doer / Try-Parse Pattern
  - Resource Pre Allocation

How Is Cloud Different?

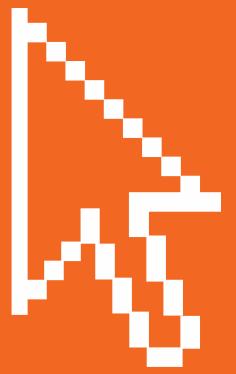
- Commodity hardware can lead to increased transient faults
- Software defined fault tolerance
  - Azure Storage – Three copies of your data
- Infrastructure as code
  - Ability to provision hardware resources

Example of Recoverable Error

- Transient fault in accessing primary storage
  - Local retry attempts exceeded the threshold
  - Point to secondary storage location
  - Retry the operation

Demo

Exception Handling in the Cloud



Logging Takes a New Meaning

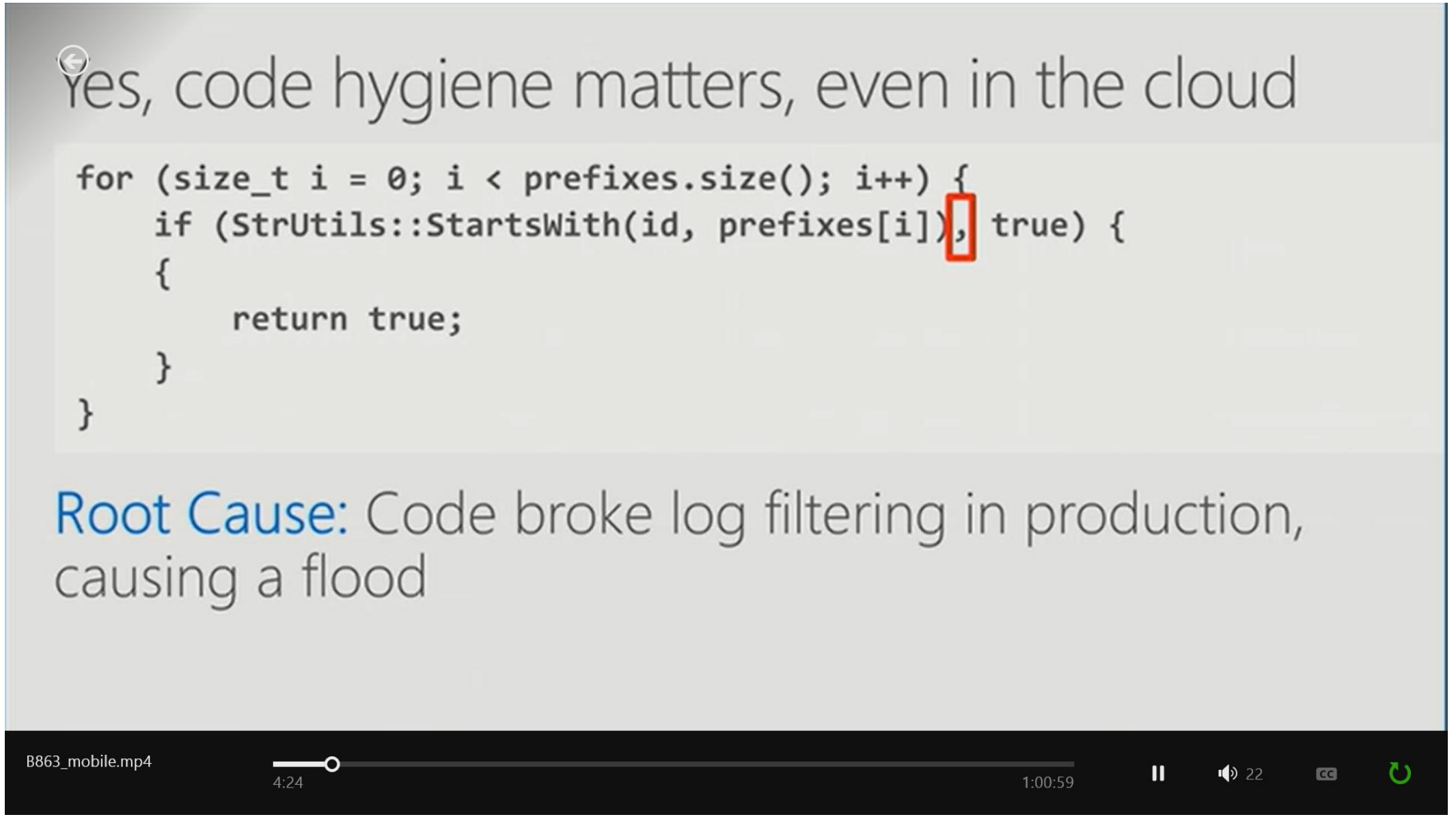
Just keep on *logging*

Logging

- Robust logging and instrumentation is critical for any successful application
- Often ETW (Event Tracing for Windows) is the place to start
- Beyond logging, consider injecting instrumentation library
- Aggregate Diagnostic Data and Analyze

How Is It Different in the Cloud

- Data scattered across multiples machines or services
- Cloud resources can disappear over time
- There is a cost association with logging and data retention
- Multi-tenancy may prevent direct access to event sources
- Use of third-party tools or SaaS needed to analyze the diagnostic data



Building Resilient Services: Learning Lessons from Azure with Mark Russinovich

Log like everybody is watching

- A little extra detail can go a long way
- Error messages should report their invalid data when possible (but don't leak sensitive info)

```
System.Reflection.TargetInvocationException: Exception has been thrown by
the target of an invocation. --->
Microsoft.ServiceModel.Web.WebProtocolException: Server Error: The service
name is unknown (NotFound)
```

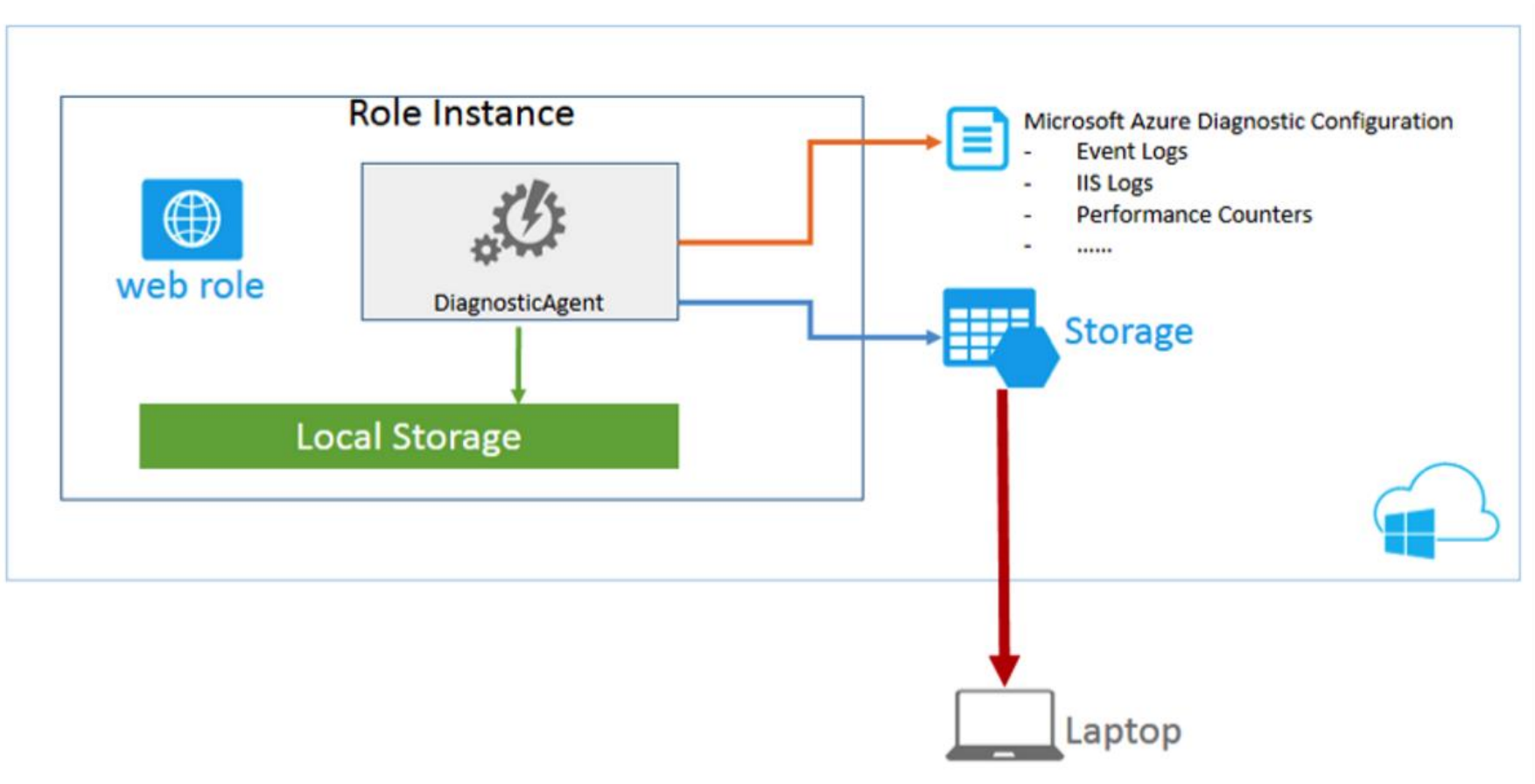
- What feature?

```
Error: The subscription is not authorized for this resource provider.
```

- Which header?

```
HTTP 400: The versioning header is not specified or was incorrect.
```

Azure Diagnostics



Demo

Logging in the Cloud

Cost Aware Computing

Utility services cost less even though they cost more

- Joe Weinman

Benefits of Cloud Computing

- Utility billing
  - Per unit cost in the cloud may be high
  - But, you only pay for what you use
- On Demand trumps capacity planning
  - Very hard to get it right
  - Avoid guesswork
  - Avoid building for peak loads (peaks are generally short lived)
- 10 laws of Cloudonomics –
  - https://gigaom.com/2008/09/07/the-10-laws-of-cloudonomics/

So… what can I can do as a developer to lower costs?

Be mindful of unused resources

- "DevOps" mindset – No more "us vs. them"
- As a developer contribute to operational efficiency
- Dissociate unused IP addresses
- Purge unused data (e.g. snapshots)
  - At the very least make it easier to detect unused resources
- Expose dynamic scaling counters
  - Dynamic scaling is as effective as the scale-out and scale-in metric!

Crush Latency

- Expect higher latency in the cloud
  - Commodity hardware
  - Shared hardware
- Build increased tolerance to latencies
- Performance efficiency often translates into lower costs to run
  - Shopzilla reported that a 5 sec speed up resulted in 50% reduction in costs
  - http://radar.oreilly.com/2009/07/velocity-making-your-site-fast.html

Tips to Minimize Latency 1/4

- Avoid blocking
  - Blocking of any sort is performance killer
  - Head-of-line blocking
- Horizontally scale from start
  - Scale-out is most cost effective approach to growing load
  - Think about scale-out from the beginning

Tips to Minimize Latency 2/4

- Loosely couple your application
  - Easier to distribute across machines
  - Match each component to cost-efficient compute unit
- Avoid runway code
  - Don’t repeat an operation endlessly if there is little chance of success
    - Poisson message handling
    - Circuit Breaker Pattern

Tips to Minimize Latency 3/4

- Avoid unnecessary tiering
  - Do you need all the tiers?
    - Or is it a pass-through tier?
      - Low utilization
      - Consider Serialization / Deserialization cost
      - Why pay for the entire VM if you're using a small % of compute
    - Or is it because of defence in depth
      - Looks carefully, PaaS service may already provide the needed security

Tips to Minimize Latency 4/4

- Worry about overall complexity of the algorithm
  - Comes into play for at high load
  - How is running time growing with load?
  - Theoretical jargon – Big-O notation
    - Worst case running time of a function
  - Example
    - If the running time of a function grows in a quadratic manner with load
      $f(n) => O(n^2)$
    - If the running time of a function grows in a linear manner with load
      $f(n) => O(n)$

Reuse++

Great programmers know what to rewrite (and **reuse**)

- Eric S Raymond

Reuse in the Cloud

- Reuse is often cited as the “holy grail” of software development
- However, cloud offers the following benefits
  - Cloud promotes homogeneity
    - Ability to reuse COTS is much higher
  - Cloud marketplace is growing
  - Framework level capabilities available as a service “Cloud OS”

SendGrid Example

```
SendGridMessage email = new SendGridMessage()
{
      From = new MailAddress(from),
      Subject = sub,
      Html = String.Format(bodyHtml, body)
};
email.AddTo(to.Split(new char[] { ',' }));
SendGrid.Web webTransport = new SendGrid.Web(new NetworkCredential(username, password));
webTransport.Deliver(email);
```

Machine Learning

- What about reusing machine developed code?
- Machine Learning (ML)
  - Define a problem and letting machine develop the code
    - Define a model and train it

Demo

Adding a recommendation engine to the reference app

Think Containers

There's just no getting around it: you're building a distributed system

– Mark Cavage

Distributed Application Challenges

- Multiple components spread across machines
  - Components have associated dependencies
    - Components' dependencies can be in conflict
  - Component deployment (initial and updates) is non-trivial
    - Scale-out model means multiple copies of each component
  - Component can have different scale characteristics
    - Write-intensive vs. read-intensive
    - Wasteful to have similar execution environments
  - Different execution environment for dev, test and prod

Micro Services

When Micro is better than Macro

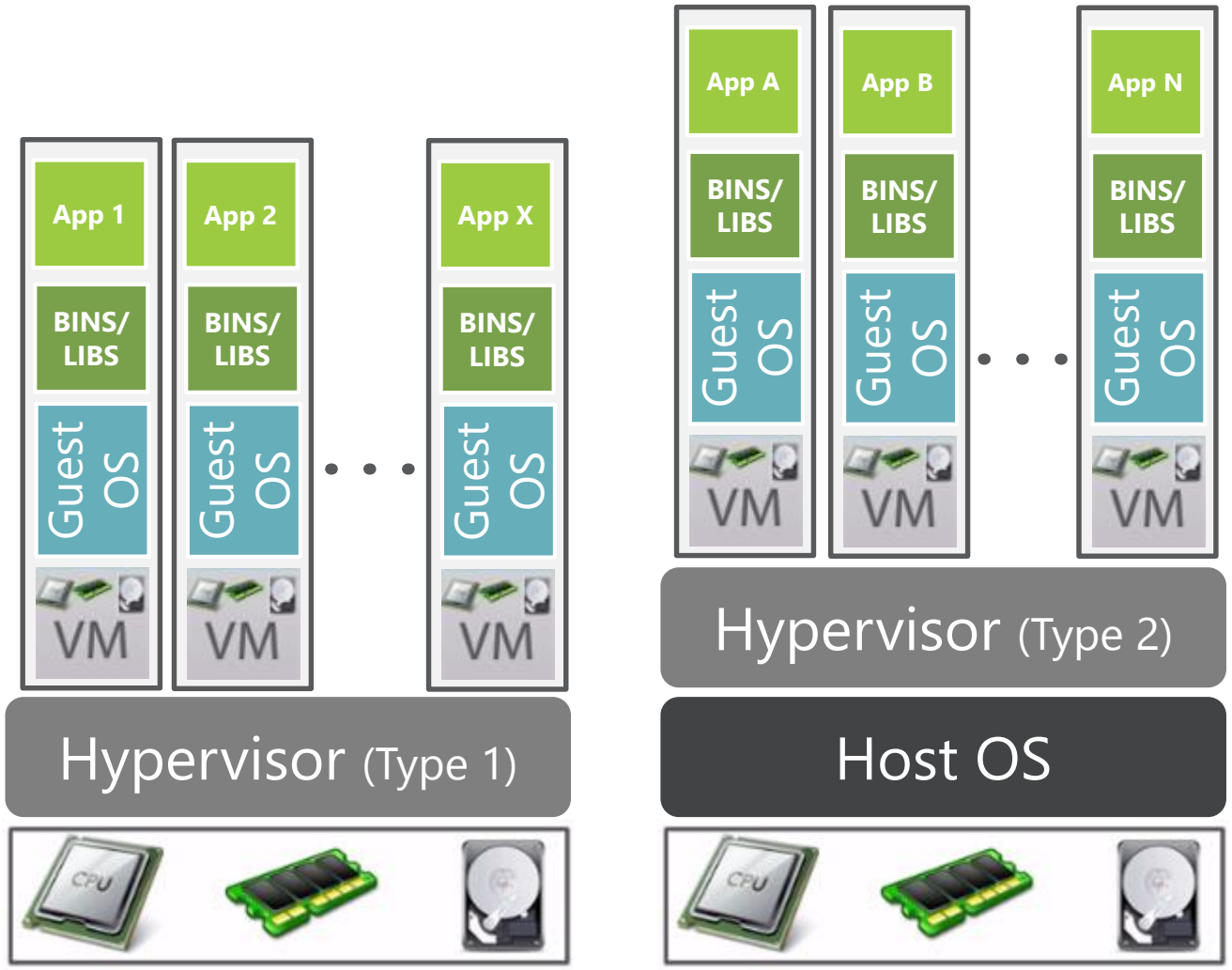
Microservices Advantages

- Single responsibility principle == strong cohesion and loosely coupled
- Failure of one microservice does not cascade to other parts
- Autonomous
- Scaled independently
- Deployed and updated independently
- Technology, language or platform agnostic
- Composable

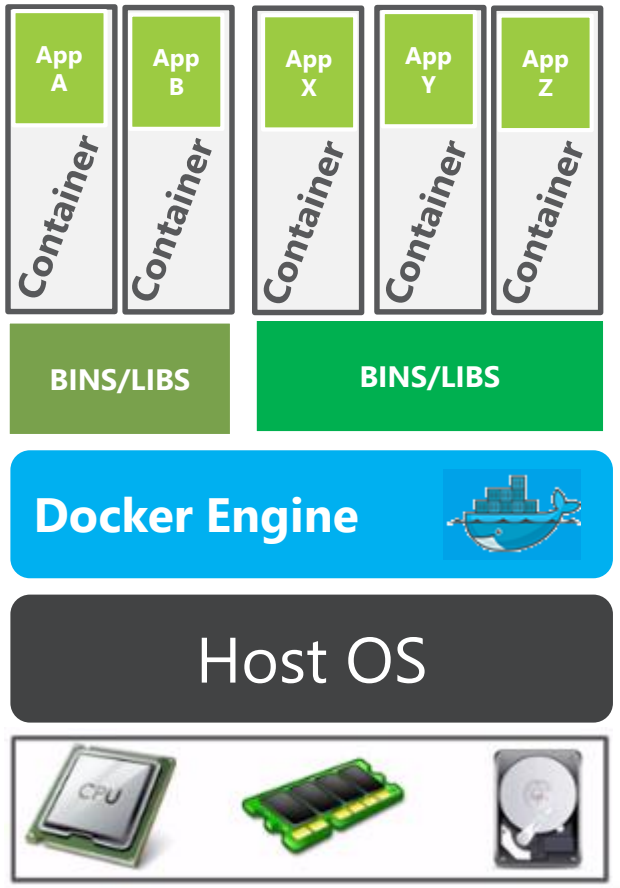
How Can Containers Help?

- Docker (name of the company and framework)
- Leverage OS specific container features (Linux and Windows)
  - Unified API and tooling to package applications
  - Holds everything needed to run an application
  - Can be started, stopped or moved
  - Based on a concept of image
    - In turn based on a collection of images made up of OS and standard components such as web server
- Docker Registries
  - Public and private

Virtual Machine Versus Docker Container



Virtualization



Containerization

Demo

Pull our Reference App from Docker Hub and start it!