Cloud and Web Applications

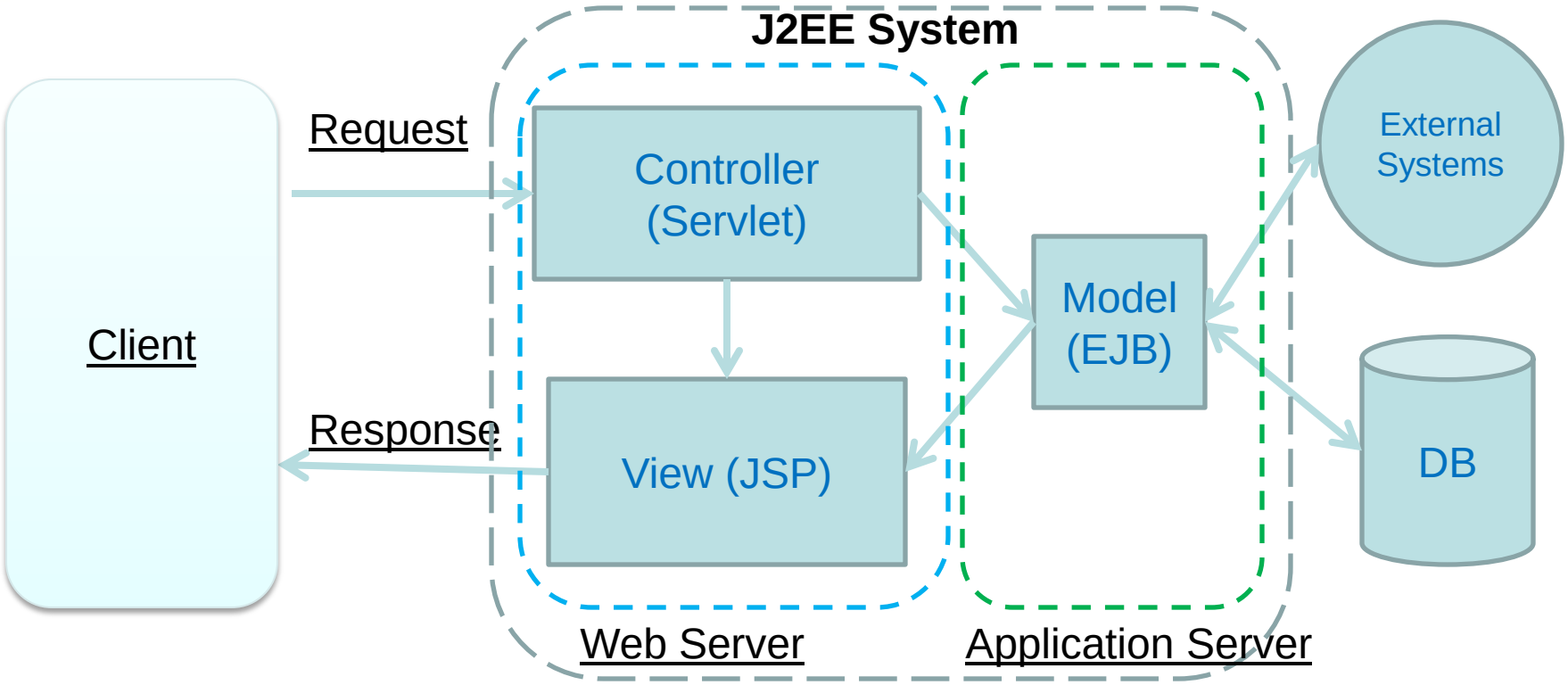
Part 3 : Server-Side Techniques

Table of Contents

- J2EE
- Node.js
- Database System
  - RDBMS
  - MongoDB
- REST
- Develop Web Apps for Cloud
- Cloud Architecture Techniques

J2EE

J2EE Technology Overview



J2EE Technology Overview (2)

- J2EE is a specification that different vendors implement
- J2EE Application Servers
  - For running full stack(Servlets, JSP, EJB)
  - jBoss, Oracle WebLogic, GlassFish, IBM Websphere …
- Java WebServers
  - For running Servlets and JSP
  - Tomcat, Jetty,
- EJB-only server
  - OpenEJB

J2EE Technology Specifications

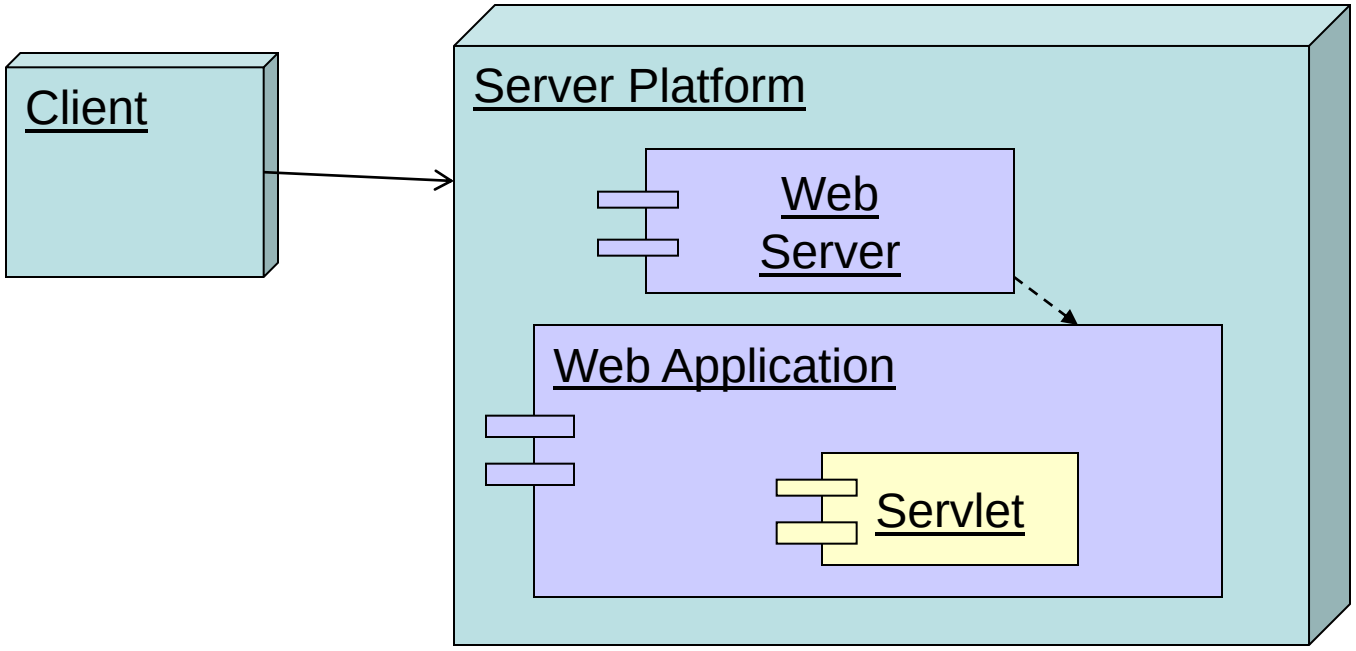
- Java API for RESTful Web Services (JAX-RS)
- Web Services 1.3 JSR109
- Java API for XML-Based Web Services (JAX-WS) 2.2 JSR224
- Java Architecture for XML Binding (JAXB) 2.2 JSR222
- Web Services Metadata for the Java Platform 2.1 JSR181
- Java API for XML-based RPC (JAX-RPC) 1.1 JSR101
- Java APIs for XML Messaging (JAXM) 1.3 JSR67
- Java API for XML Registries (JAXR) 1.0 JSR93
- Java Servlet 3.0 JSR315
- JavaServer Faces (JSF) 2.0 JSR314
- JavaServer Pages (JSP) 2.2 JSR245
- Expression Language (EL) 2.2 JSR245
- JavaServer Pages Standard Tag Library (JSTL) 1.2 JSR52
- Enterprise JavaBeans (EJB) 3.1 JSR318 Lite
- Java Persistence API (JPA) 2.0 JSR317
- Contexts and Dependency Injection for Java 1.0 JSR299
- Dependency Injection for Java 1.0 JSR330
- Common Annotations for the Java Platform 1.1 JSR250
- Java Message Service API (JMS) 1.1 JSR914
- Java Transaction API (JTA) 1.1 JSR907
- JavaMail API 1.4 JSR919
- Java Authentication Service Provider Interface for Containers (JASPIC) 1.0 JSR196
- Java Authorization Service Provider Contract for Containers (JACC) 1.4 JSR115
- …

Web Technologies in J2EE

- Java Servlet 3.0
- JavaServer Faces 2.0
- JavaServer Pages 2.2/Expression Language 2.2
- Standard Tag Library for JavaServer Pages (JSTL)
- Debugging Support for Other Languages 1.0

Java Servlets

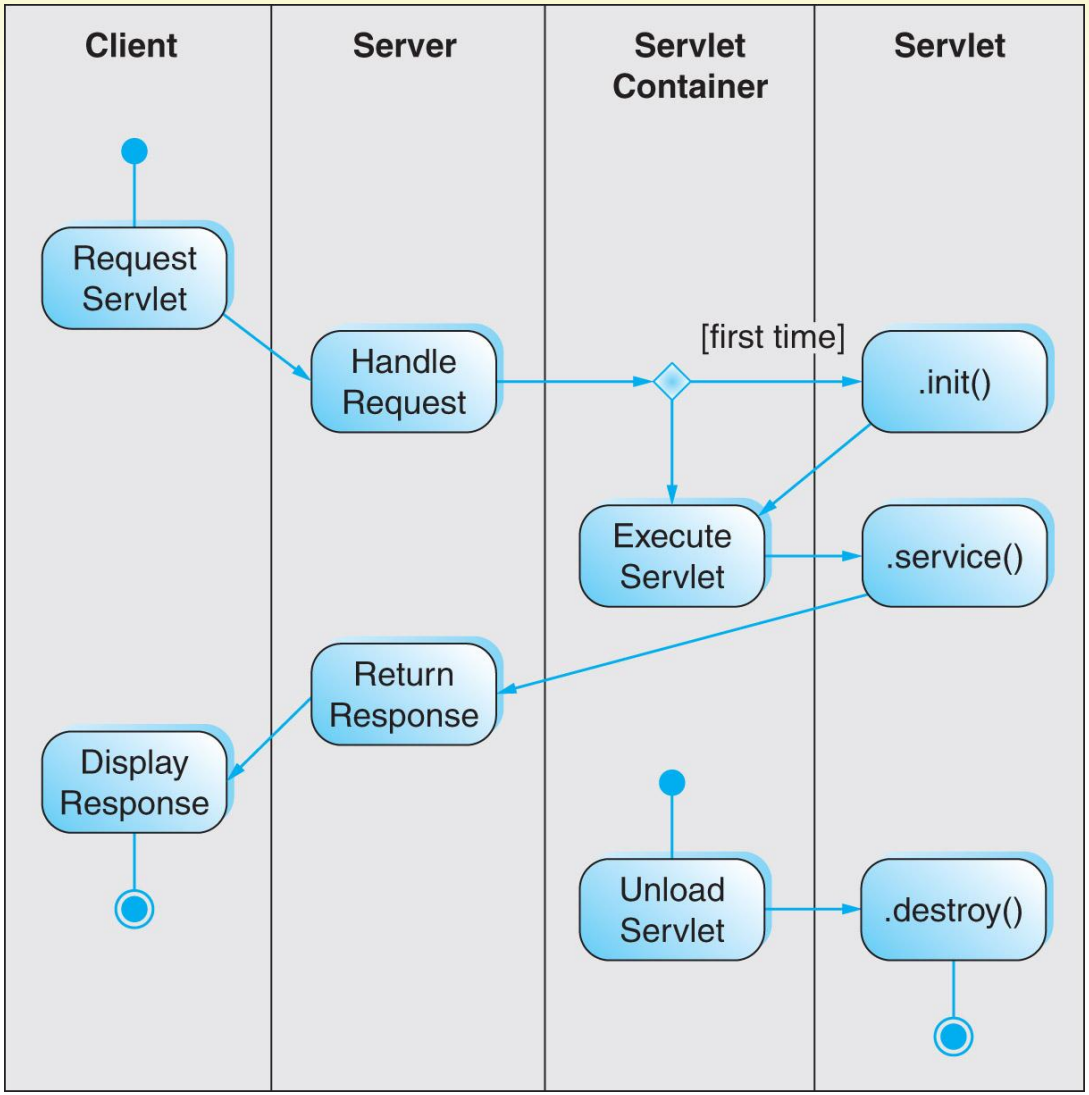
- A servlet is a Java program that is invoked by a web server in response to a request.
- Hosted by a servlet container



The web server handles the HTTP transaction details

Example: Apache Tomcat, both a web server and a servlet container

Servlet Operation



Servlet Example

- This servlet will say "Hello!" (in HTML)

```
package servlet;
import javax.servlet.http.*;

public class HelloServlet extends HttpServlet {
  public void service(HttpServletRequest req,
      HttpServletResponse res) throws IOException {
    PrintWriter htmlOut = res.getWriter();
    res.setContentType("text/html");
    htmlOut.println("<html><head><title>" +
      "Servlet Example Output</title></head><body>" +
      "<p>Hello!</p>" + "</body></html>");
    htmlOut.close();
  }
}
```

Servlet Configuration

- The web application configuration file, web.xml, identifies servlets and defines a mapping from requests to servlets

An identifying name for the servlet (appears twice)

```
<servlet>
  <servlet-name>HelloServlet</servlet-name>
  <servlet-class>servlet.HelloServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>HelloServlet</servlet-name>
  <url-pattern>/hello</url-pattern>
</servlet-mapping>
```

The servlet's package and class names

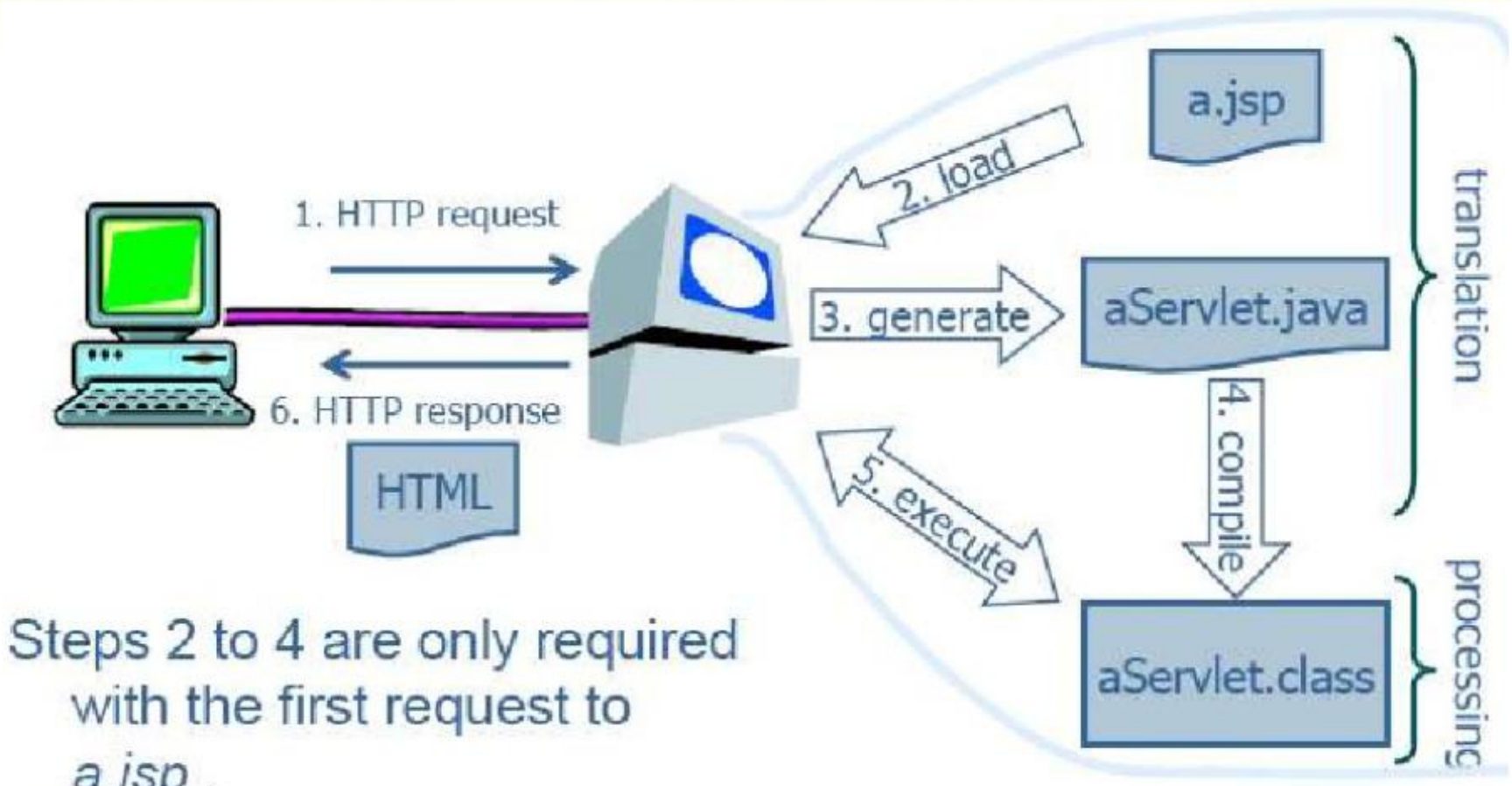
The pathname used to invoke the servlet (relative to the web application URL)

Java Server Pages (JSP)

- HTML document with embedded Java code
- Translated into Java servlets when first used
- Allow static HTML content to be separated from dynamic content

```
<%@ page %>
<html>
<head><title>Example Page</title></head>
<body>
<h1>Example Page</h1>
<%
  String name = req.getParameter("name");
%>
<p>Hello, <%= name %></p>
...
</html>
```

Execution of JSP



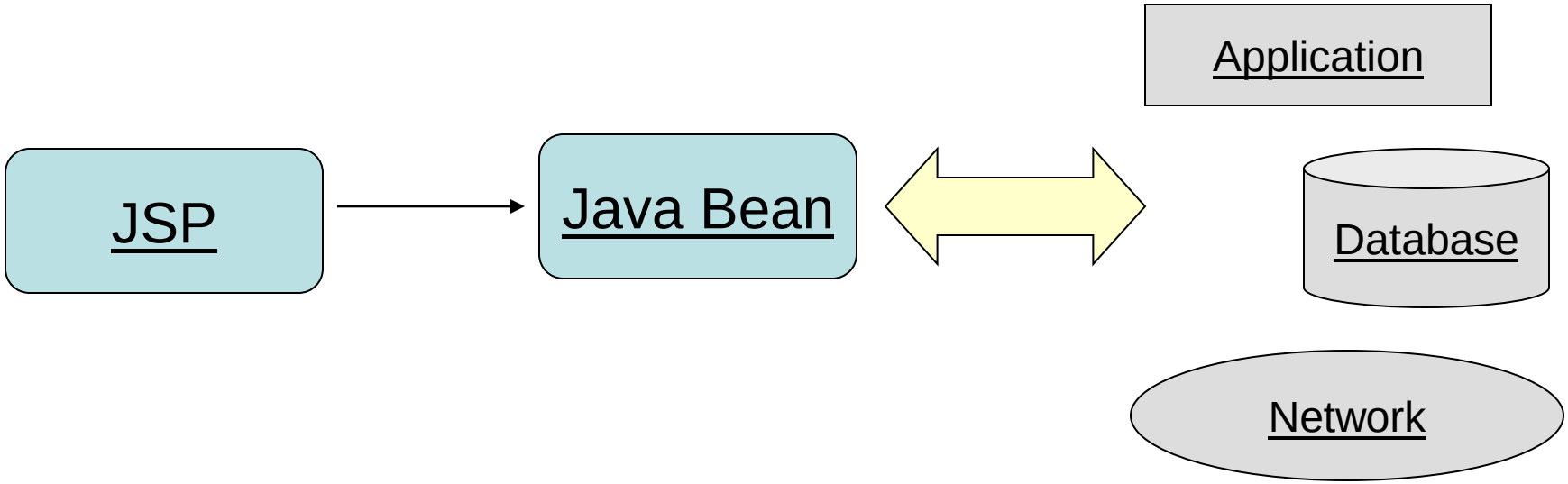
Steps 2 to 4 are only required with the first request to *a.jsp* .

Java Beans

- A Java Bean is a Java class that can be invoked from a JSP
- This allows complex processing required by a JSP to be separated from the JSP
- A JSP can instantiate an instance of a bean class and invoke its methods

Java Bean Uses

- Java Beans are typically used to encapsulate complex application logic, including calculations, database access, or network access



JSP and Java Bean Example

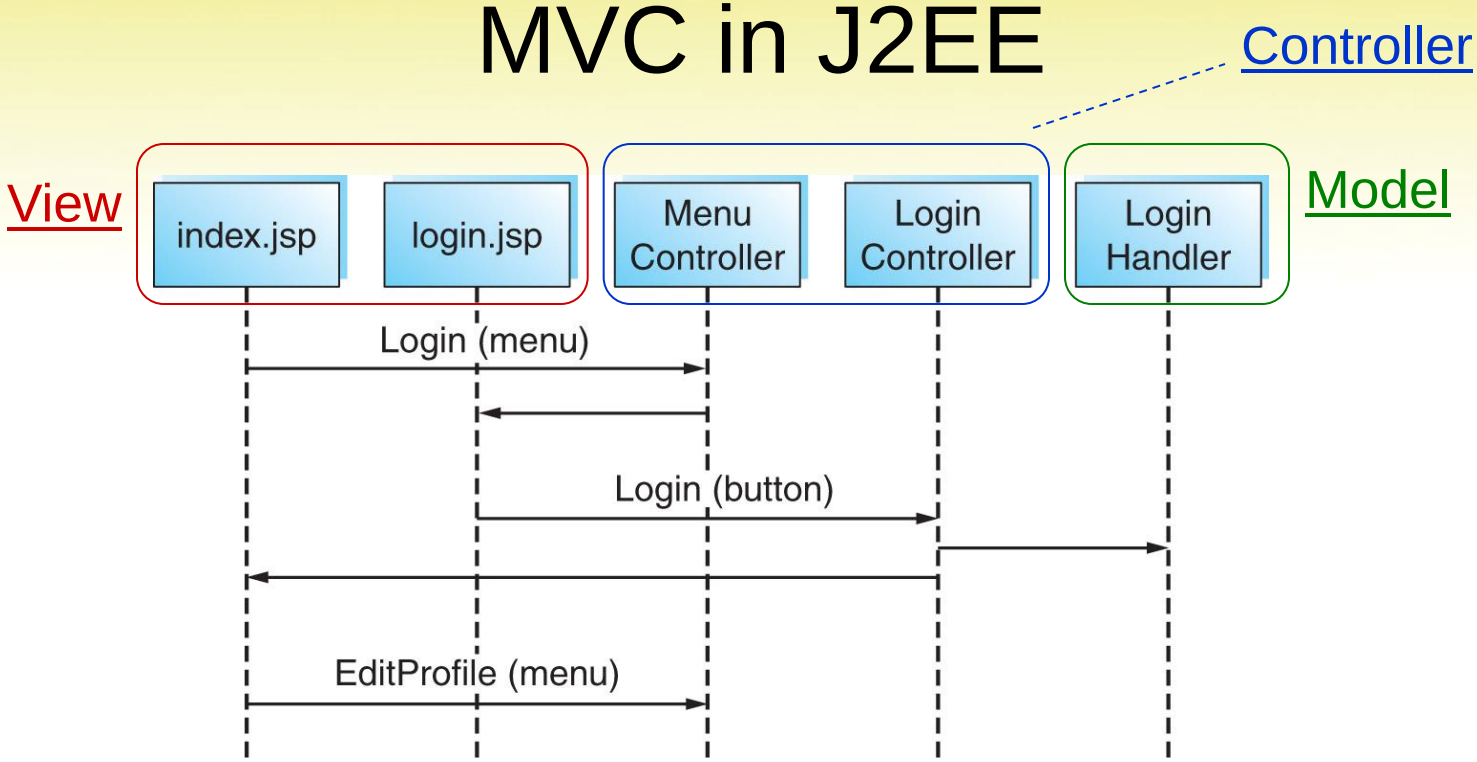
- Java Bean

```
package bean;
public class AdderUtility {
  public int add(int a, int b) {
    return a + b;
  }
}
```

- JSP

```
<%@ page %>
<jsp:useBean id="adder" class="bean.AdderUtility"
   scope="session" />

<%
  int a, b, sum;
  …
  int sum = adder.add(a, b);
%>
```

MVC in J2EE



1. User requests "login" option; Controller sets "login.jsp" as the next view.
2. User enters credentials; Controller invokes LoginHandler to handle transcation
3. etc.

Traditional J2EE Users

- Examples of use
  - Insurance companies
  - Online banks
  - Manufacturing industries
  - Public sector

'Cobol of the 21st century'

New J2EE Techniques

- Many languages with seamless integration: you can share libraries, code, classes, etc.
  - Groovy
    - Object-oriented
    - Inspired by Java, Python, Ruby, Smalltalk
  - Scala
    - Functional, object-oriented
    - ”Cutting away Java’s syntactic overhead, adding power”
    - Inspired by Java, Scheme, Erlang, Haskell, Lisp
  - JRuby: implementation of Ruby on JVM
  - Jython: implementation of Python on JVM

Extensions to Core J2EE

- Grails (Groovy on Rails)
  - Interpreted script compiled to Java bytecode at runtime
  - Supports writing Java code directly
  - Java libraries, JPA models etc directly available
- Spring / Spring MVC
  - Abstraction, no need to work directly with Servlets etc
    - XML-based configuration
  - IoC/DI
  - Java EE 6 based on Spring
- JSF - Java Server Faces
  - Notation for generating JSF-pages (View), which communicate with managed beans(Controller)
- Struts
  - Custom tag libraries
  - XML-based

J2EE Trends

- Moving from single-tier or two-tier to multi-tier architecture
- Moving from monolithic model to object-based application model
- Moving from application-based client to HTML-based client

“Trendy” J2EE users: LinkedIn

- Started with Java platform, using Java EE and extensions
  - Spring Framework
  - Grails
- Now utilizing also Scala and JRuby
  - Scala for back-end processing
  - JRuby for integration interfaces

“Trendy” J2EE users: Twitter

- Started with Ruby on Rails
- Now using Java and Scala in back-end processing
  - Why?
    - Scalability and Performance
    - SOA
    - Encapsulation (re-use, maintenance)

“Trendy” J2EE users: Others

- Google, Amazon and many others use J2EE
- What about Facebook?
  - Writing PHP, which quickly lead to serious performance issues
  - Started compiling PHP to C++
  - Are now investigating using PHP on JVM

NODE.JS

Introduction to Node.js

"Node.js is a platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices."

- nodejs.org

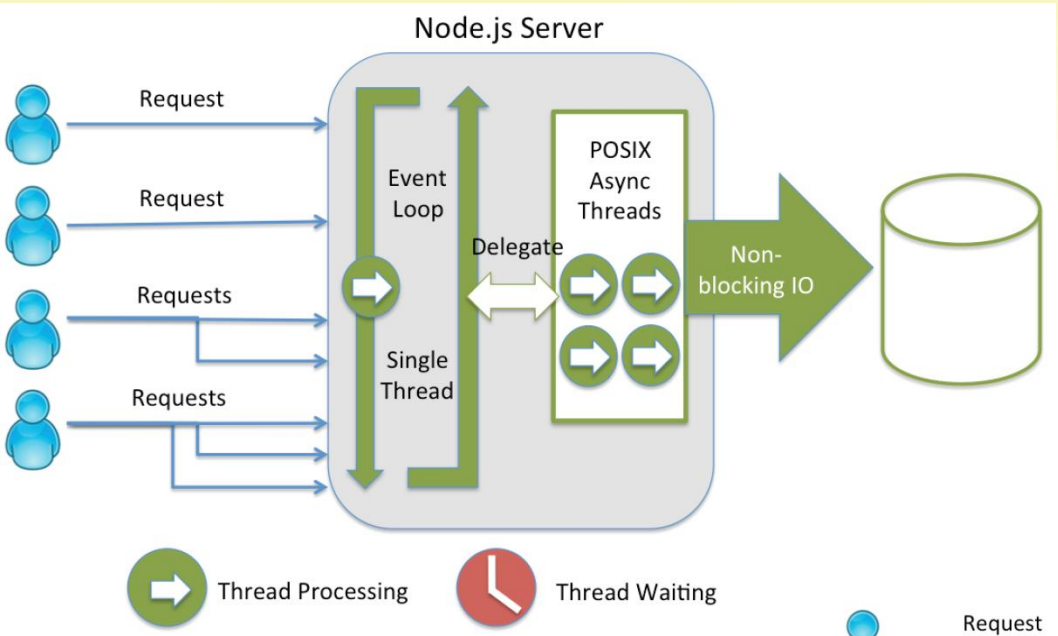
Preface

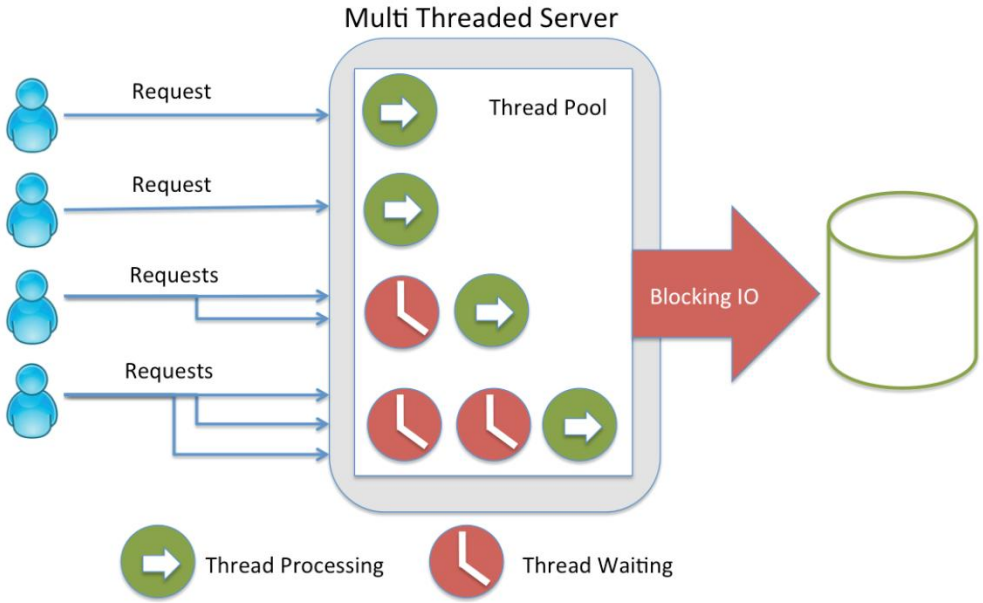
- Developed by Ryan Dahl pada tahun 2009
- Server Side JavaScript
- Built on Google's V8
- An environment for developing high performance web services
- Using event-driven, asynchronous I/O to minimize overhead and maximize scalability.
- The goal is to provide an easy way to build scalable network servers

Threads vs. Event-Driven

Threads	Asynchronous Event-driven
Lock application / request with listener-workers threads	only one thread, which repeatedly fetches an event
Using incoming-request model	Using queue and then processes it
multithreaded server might block the request which might involve multiple events	manually saves state and then goes on to process the next event
Using context switching	no contention and no context switches
Using multithreading environments where listener and workers threads are used frequently to take an incoming-request lock	Using asynchronous I/O facilities (callbacks, not poll/select or O_NONBLOCK) environments

Threads vs. Event-Driven (2)





Why Node.js Use Event-based?

In a normal process cycle the web server while processing the request will have to wait for the I/O operations and thus blocking the next request to be processed.

Node.JS process each request as events, The server doesn't wait for the IO operation to complete while it can handle other request at the same time.

When the I/O operation of first request is completed it will call-back the server to complete the request.

Node.js Potential Wins

- JavaScript is the popular language of the web
- Web applications spend most of their time doing I/O
- Can implement the same programming language on client and server. Code can be migrated between server and client more easily
- Common data formats (JSON) between server and client
- Common software tools, testing or quality reporting tools for server and client
- Node.js ships with a lot of useful modules, so you don't have to write everything from scratch.

- Thus, Node.js is really two things: a runtime environment and a library

Node.js vs. Apache

Platform	Number of request per second
PHP (via Apache)	3187,27
Static (via Apache)	2966,51
Node.js	5569,30

Node.js vs. PHP + Nginx

Platform	Static (number of request per second)	Query MySQL (number of request per second)
PHP + Nginx	3624	1293
Node.js	7677	2999

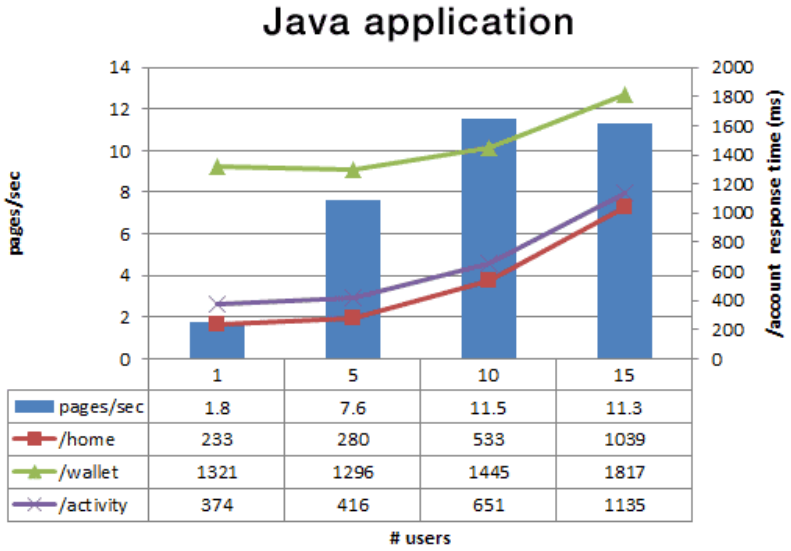
-- by Snoopyxd

Node.js at PayPal

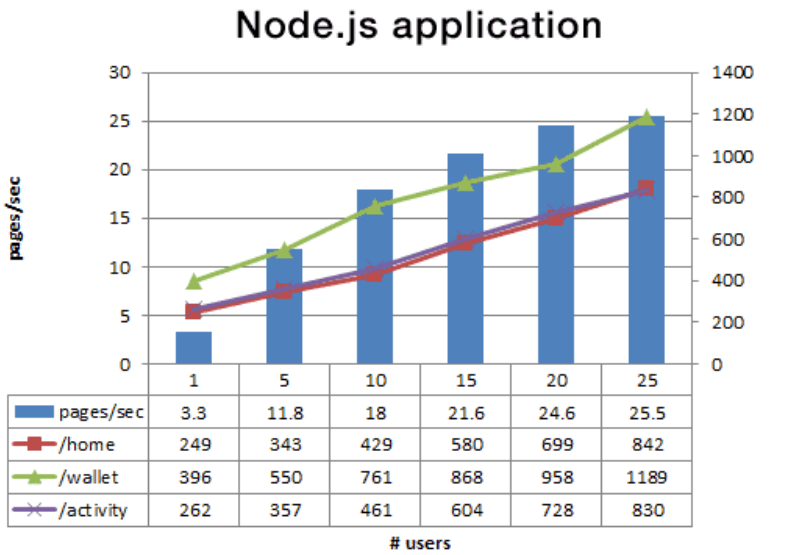
- PayPal's web applications are moving away from Java and onto JavaScript and Node.js.
- Comparing with Java, the Node.js app was:
  - Built almost **twice as fast with fewer people**
  - Written in **33% fewer lines of code**
  - Constructed with **40% fewer files**

From https://www.paypal-engineering.com/2013/11/22/node-js-at-paypal/

Performance of PayPal App.



	1	5	10	15
pages/sec	1.8	7.6	11.5	11.3
/home	233	280	533	1039
/wallet	1321	1296	1445	1817
/activity	374	416	651	1135



	1	5	10	15	20	25
pages/sec	3.3	11.8	18	21.6	24.6	25.5
/home	249	343	429	580	699	842
/wallet	396	550	761	868	958	1189
/activity	262	357	461	604	728	830

- Double the requests per second vs. the Java application.
- 35% decrease in the average response time for the same page. This resulted in the pages being served 200ms faster.

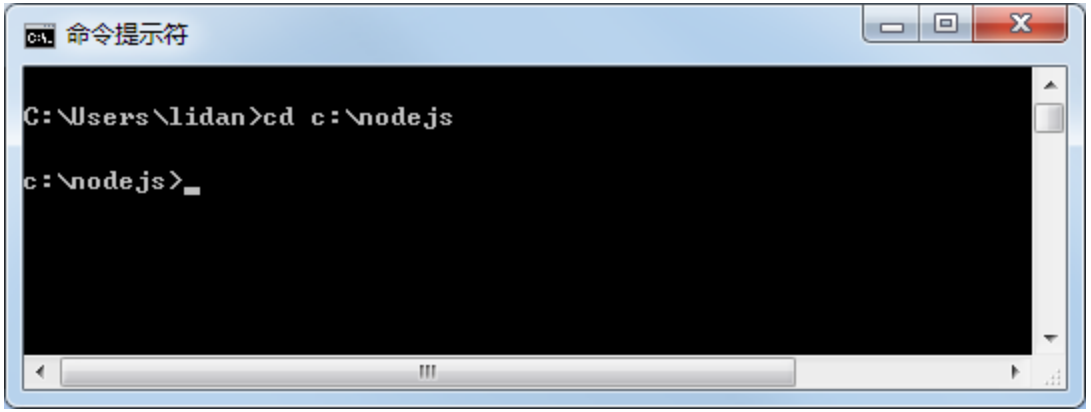
Lab : Open Command Prompt

Cmd is the command line interface tool used to execute the various commands.

- Win XP: **Start button** -> **Run ->** type cmd.exe, press enter.
- Win 7: **Start button** -> type cmd in the search box -> Click on **cmd** the search results listing.
- Win 8: Swipe to **Apps** screen -> locate *Windows System* -> click on **Command Prompt**
- Win 10: Press win+R, type cmd.exe in the prompt box, press enter.

Basic cmd commands

- dir : display a list of files and folders
- cd <folder> : change directory - move to a specific *folder*



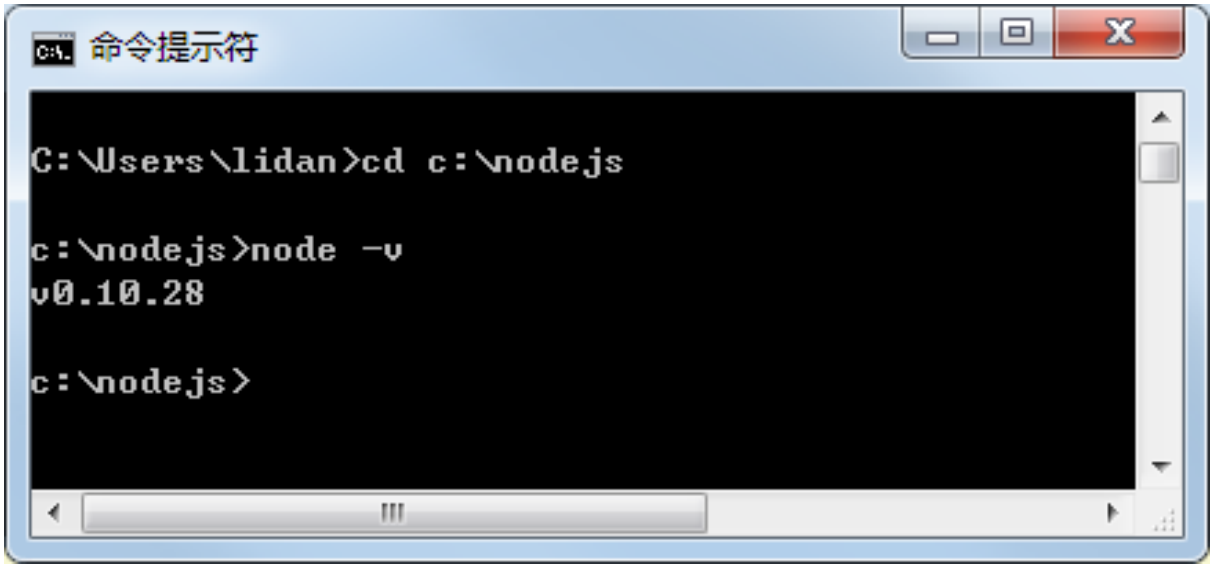
- md <name> : create a new folder with <*name*> in the current folder.
- ping <ip> : test a network connection to the ip

Lab : Install NodeJS

- Download and install node.js : http://nodejs.org/
  - Click ‘Install’ or go to the ‘Downloads’ page
  - Once downloaded, run the installer
  - Install Node.js in directory `c:\>nodejs`

Test the Installation

- Open a cmd, go to the **`nodejs`** directory
- Type **`node -v`** in command line
- If the version information is shown, Node.js is correctly installed.

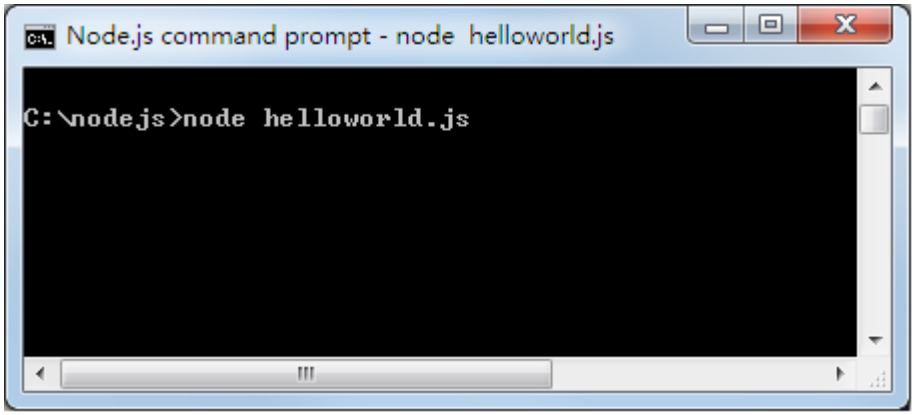


A Basic HTTP Server

- Create **helloworld.js** file in the nodejs folder as the follows

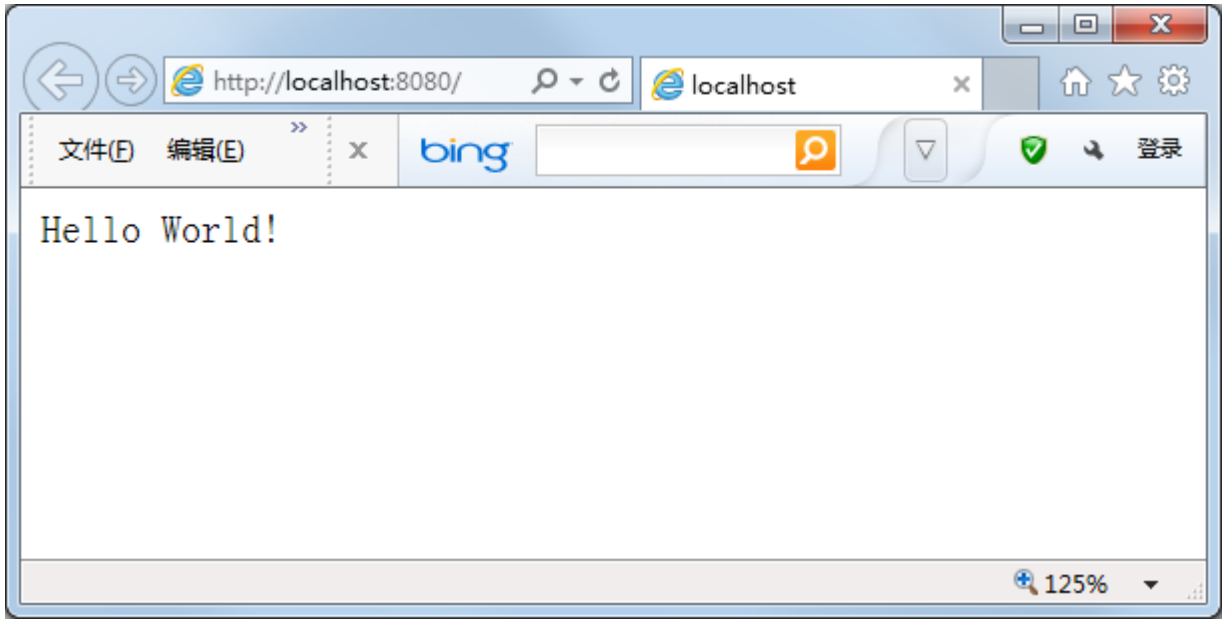
```
var http = require("http");
http.createServer(function(request, response) {
    response.writeHeader(200, {"Content-Type": "text/html"});
    response.write("Hello World!");
    response.end();
}).listen(8080);
console.log("Server has started.");
```

- Execute the js file in Node.js:



Test the HTTP Server

- Open your browser and access `http://localhost:8080`. This should display a web page that says “Hello World”.



Analyzing the HTTP Server

requires the "http" module from Node.js library and assigns it to variable http

```
var http = require("http");

http.createServer(function(request, response) {
    response.writeHeader(200, {"Content-Type": "text/html"});
    response.write("Hello World!");
    response.end();
}).listen(8080);

console.log("Server has started.");
```

the parameter of the createServer *: an anonymous function to callback when a http request is arrived*

Invokes method listen *of the object , listen to a http port*

calls function createServer *of the http module, returns an object*

Another Version of the HTTP Server

```
var http = require("http");

function onRequest(request, response) {

  console.log("Request received.");

  response.writeHead(200, {"Content-Type": "text/plain"});
  response.write("Hello World");
  response.end();
}

http.createServer(onRequest).listen(8080);

console.log("Server has started.");
```

DATABASE SYSTEMS

RDBMS

- A Relational Database Management System (RDBMS) provides storage and access for structural data in web applications

- Examples: MySQL, SQL Server, Oracle, PostreSQL

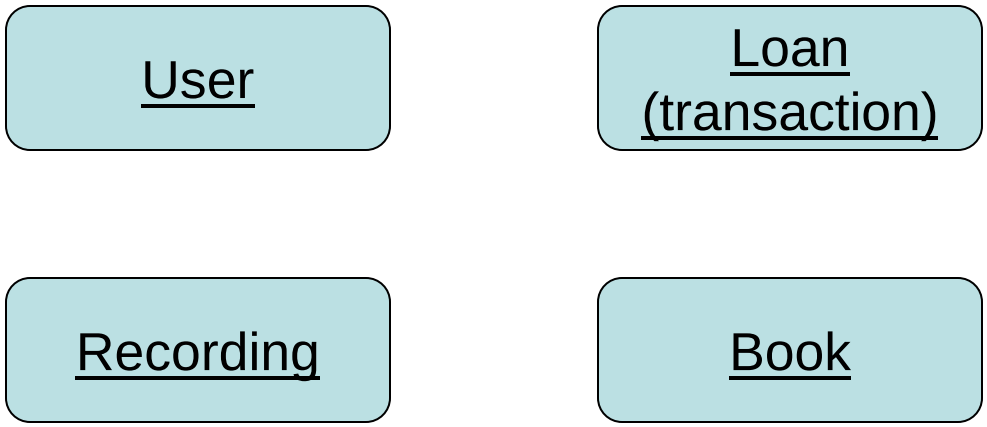
Other Types of Databases

- Hierarchical database
  - organizes data in a tree-like structure
  - defines a parent/child relationship
  - each parent can have many children but each child has only one parent
  - folder or directory structure for storing files on a computer is an example of a hierarchical data structure
- NoSQL- promotes the use of non-relational databases and does not require fixed table schemas as with the relational model

Relational Database Structure

- A database includes one or more tables
  - Each table represents one *type* of entity

- Example: Tables in a Library Database



Relational Database Structure (2)

- Each table field represents an entity attribute
- Each table row represents one entity

Car table:

Year	***Make***	***Model***	***Color***
1973	Volkswagen	Jetta	Red
1992	Ford	Aerostar	Blue
2004	Chevrolet	Suburban	Black

row

field

Structured Query Language (SQL)

- Development since 1970 by IBM
- A standard language for creating and maintaining relational databases
- SQL statement types:
  - *data definition*: create databases and tables
  - *data manipulation*: add, modify, delete data
  - *data control*: set access permissions

Basic SQL Statements

- Data definition
  - CREATE, DROP
- Data manipulation
  - SELECT, INSERT, UPDATE, DELETE
- Data control
  - GRANT, REVOKE

CREATE

- Create a database or a table

```
CREATE DATABASE mydata

CREATE mydata.player (
  playerNr int PRIMARY KEY,
  name VARCHAR(30),
  isCurrent BOOLEAN NOT NULL)
```

Example table creation

```
create table student
(
    id int,
    name varchar(255),
    major char(4),
    gender char(1),
    dob date,
    constraint student_pk primary key (id)
);
```

why did we specify these attribute types?

student_pk is an arbitrary name

Basic SQL Data Types

- INTEGER
- DECIMAL(T, R)
  - T=total digits, R=right digits (after '.')
- FLOAT

- CHAR(N) N characters
- VARCHAR(N) up to N characters

- BOOLEAN

- DATE
- TIME

DROP

- DROP can be used to delete an entire database or a table

```
DROP mydata

DROP mydata.player
```

SELECT

- SELECT retrieves data from a database

```
SELECT field-list FROM database.table
  WHERE condition
  ORDER BY field-list
```

  - *field-list* is a comma-separated list of fields from the named table (* means "all fields")
  - *condition* is a Boolean condition using field names and/or constants

SELECT Example

- Example

```
SELECT * FROM mydata.player

SELECT playerNr, name FROM player
  WHERE isCurrent=TRUE

SELECT playerNr, name, status FROM player
  WHERE playerNr >= 90001
  ORDER BY status, name
```

INSERT

- INSERT adds a new row to a table

```
INSERT INTO player
  VALUES (23752, 'Jane Doe', TRUE)
```

UPDATE

- UPDATE changes one or more rows

```
UPDATE database.table
  SET field-assignment-list
  WHERE condition

UPDATE player
  SET isCurrent=TRUE
  WHERE playerNr=33256
```

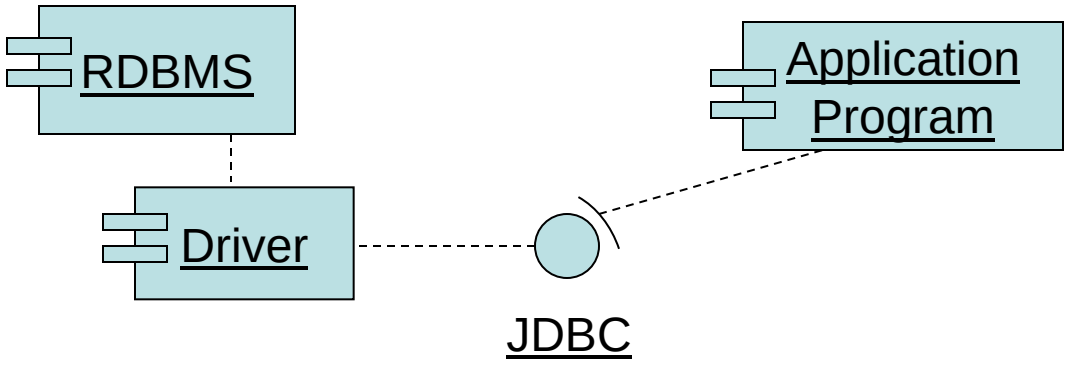
DELETE

- DELETE removes one or more rows from a table

```
DELETE FROM database.table
  WHERE condition

DELETE FROM player
  WHERE playerNr=33523
```

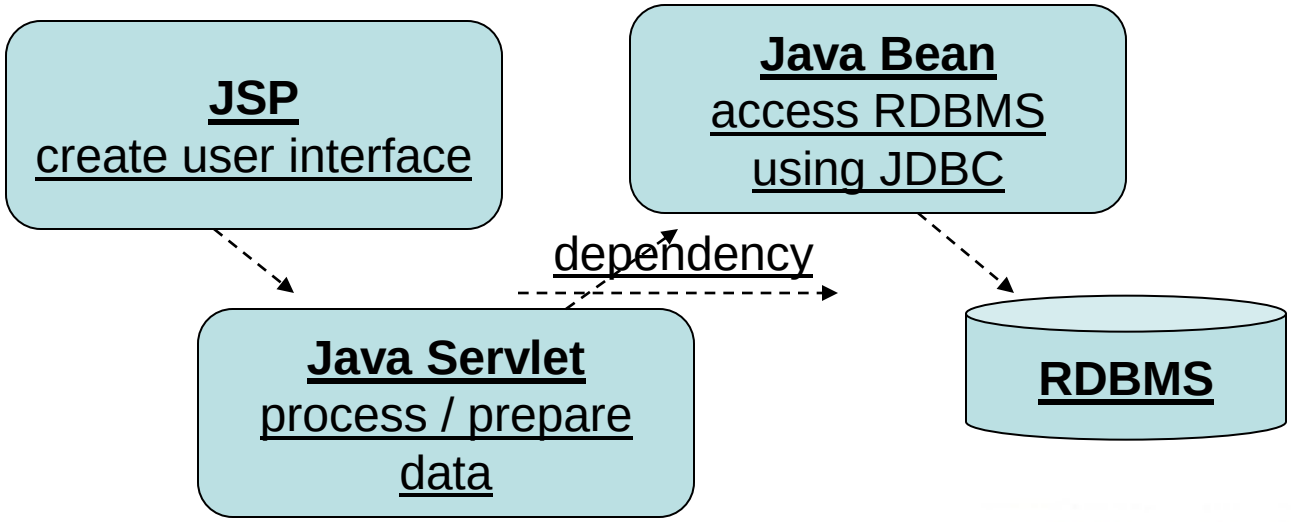
JDBC

- Java Database Connectivity (JDBC) is a Java API that allows Java programs to interact with relational database management systems
- Interaction also requires a database driver, which translates JDBC commands to procedure calls on the RDBMS



JDBC Design

- An effective design for database access:
  - JSP: user interface presentation
  - Java Servlet: application logic
  - Java Bean: database access (JDBC)



NoSQL Databases

- A NoSQL database provides a mechanism for storage and retrieval of data that employs less constrained consistency models than traditional relational databases.
- NoSQL databases only support eventual consistency which is a consistency model used in distributed computing that informally guarantees that, if no new updates are made to a given data item, eventually all accesses to that item will return the last updated value.

NoSQL Databases (2)

- NoSQL databases are often highly optimized key–value stores intended for simple retrieval and appending operations, with the goal being significant performance benefits in terms of latency and throughput.
- Key–value stores allow the application to store its data in a schema-less way.
  - The data could be stored in a datatype of a programming language or an object. Because of this, there is no need for a fixed data model.

NoSQL Market Data

2014 **8%**

% NoSQL Enterprise Adoption

2010 **5%** 2015 **20%**

RedMonk

"MongoDB is the new MySQL."

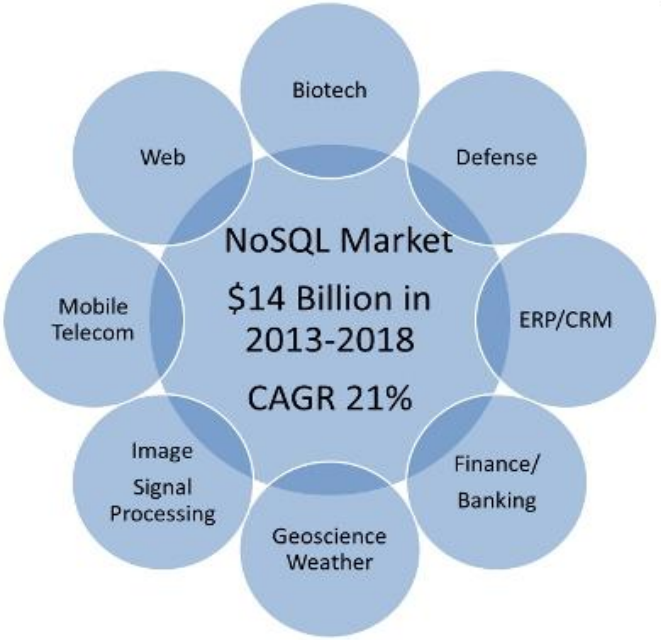
FORRESTER®

"Current adoption of NoSQL in enterprises is 4% in 2010. expected to double by 2014, and grow to 20% by 2015. 10% of existing apps could move to NoSQL. [Sept 2011]

Gartner

"NoSQL is in its infancy, many immature with only community support and limited tool capability; however, expect to see a few used more widely during the next 5 years."

- 2011: Market Research Media noted worldwide NoSQL market was expected to reach ~$3.4B by 2018, generating $14B from 2013-2018 with a CAGR of 21%
- Comparison: data implies NoSQL market ~$895M
  - MySQL in 2011 ~$100M



RDBMS vs. NoSQL

- RDBMS
  - Transational guarantee, complex functionality, Data consistency, correctness
  - Hard to scale
- NoSQL
  - Simple key-value data model, schema free, simple functionality, relaxed consistency
  - Horizontal scalability, high availability
- NewSQL = RDBMS + NoSQL

MongoDB Overview

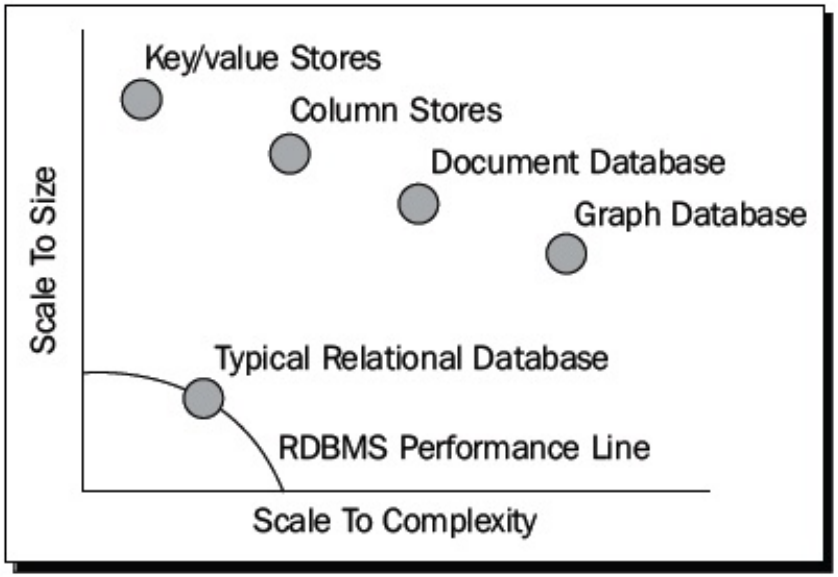
- Document oriented, not table/row oriented
- Collection of binary JSON (BSON) documents
- Schemaless
- No relations or transactions native in database
- Scalable and high-performance
- Fast In-Place Updates
- Map/Reduce
- Full index support
- Servers for all major platforms
- Drivers for all major development environments
- Free and open-source, but also commercial support

Why MongoDB

- Rapid Application Prototyping
  - Schema rides as objects inside the JSON Script, not as a separate item.
    - Schemas change quite quickly in modern apps as business needs change.
    - Apps have shorter production lives and even shorter life-cycles as a result
- Scalability
  - Automatic data rebalancing across cheap, inexpensive commodity network hardware during the shard operation.
- Large amount of customers worldwide.
- Free basic edition
- Many related free software downloads

Use cases

- High performance and scalable applications
- Most web applications where you would previously use SQL



Do not use for:

- Transaction-critical applications

BSON - Data Type in MongoDB

- BSON is the binary representation of JSON.
- MongoDB drivers send and receive in BSON form.

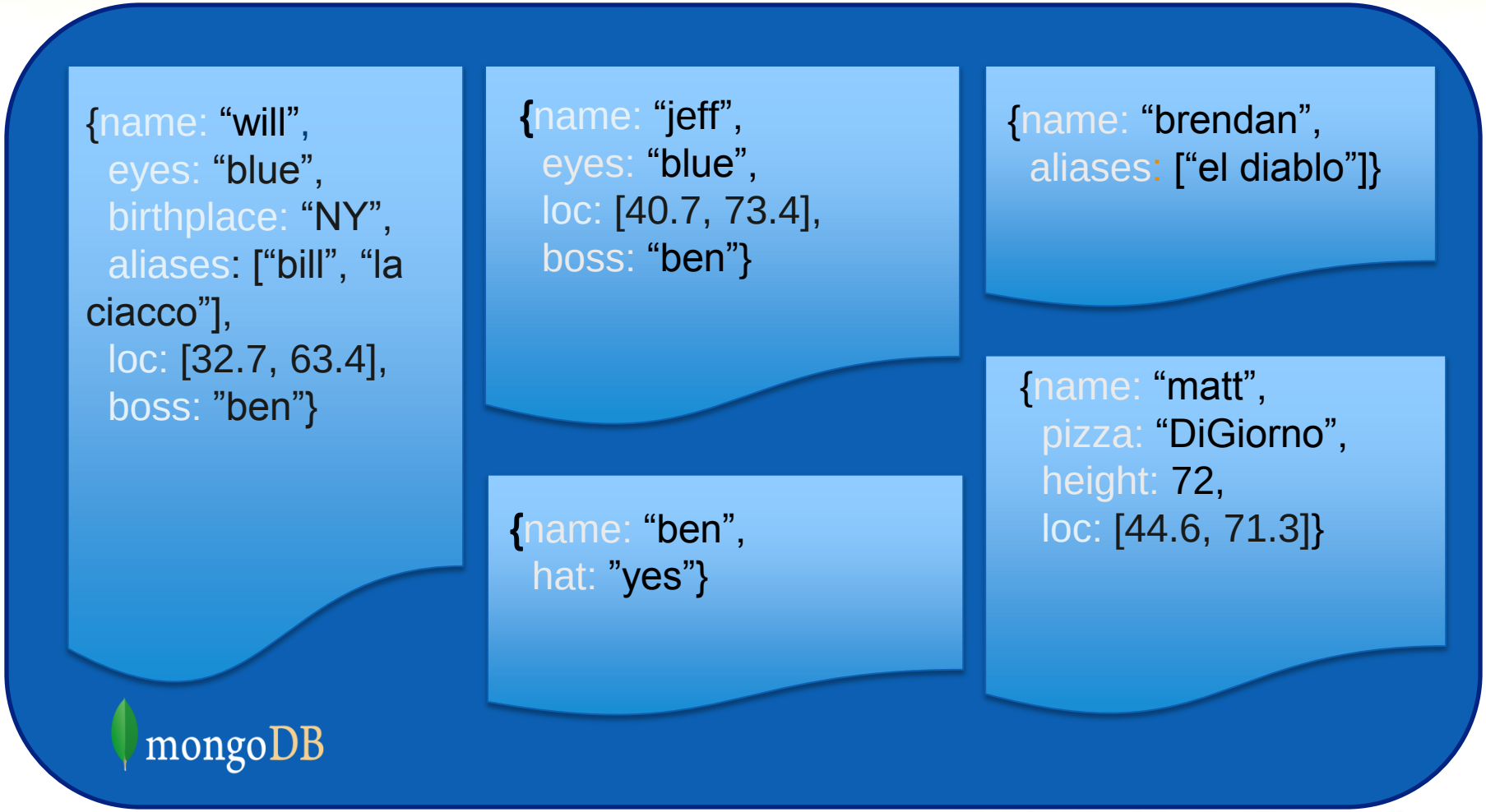
```
{"hello":
"world"}        →   "\x16\x00\x00\x00\x02hello\x00
                     \x06\x00\x00\x00world\x00\x00"

                    "\x31\x00\x00\x00\x04BSON\x00\x26\x00
{"BSON":             \x00\x00\x020\x00\x08\x00\x00
["awesome",     →    \x00awesome\x00\x011\x00\x33\x33\x33\x33\x33\x33
5.05, 1986]}         \x14\x40\x102\x00\xc2\x07\x00\x00
                     \x00\x00"
```

Terminology and Concepts

RDBMS		MongoDB
Database	⇨	Database
Table, View	⇨	Collection
Row	⇨	Document (JSON, BSON)
Column	⇨	Field
Index	⇨	Index
Join	⇨	Embedded Document
Foreign Key	⇨	Reference
Partition	⇨	Shard

Schema Free

- MongoDB does not need any pre-defined data schema
- Every document could have different data!



CRUD

- Create
  - db.collection.insert(<document>)
  - db.collection.save(<document>)
  - db.collection.update(<query>, <update>, { upsert: true })
- Read
  - db.collection.find(<query>, <projection>)
  - db.collection.findOne(<query>, <projection>)
- Update
  - db.collection.update(<query>, <update>, <options>)
- Delete
  - db.collection.remove(<query>, <justOne>)

Example-Querying

```
> db.users.findOne({age:39})
> db.users.find({'last': ‘Doe'})
// retrieve all users order by last_name:
> db.users.find({}).sort({last: 1});
```

```
-       {"_id" : ObjectId("5114e0bd42…"),
        "first" : "John",
        "last" : "Doe",
        "age" : 39,
       "interests" : [
              "Reading",
              "Mountain Biking ]
       "favorites": {
             "color": "Blue",
             "sport": "Soccer"} }
```

Advanced Querying

```
{ name: "Joe", address: { city: "San Francisco", state:
   "CA" } , likes: [ 'scuba', 'math', 'literature' ] }

// field in sub-document:
db.persons.find( { "address.state" : "CA" } )

// find in array:
db.persons.find( { likes : "math" } )

// regular expressions:
db.persons.find( { name : /acme.*corp/i } );

// javascript where clause:
db.persons.find("this.name != 'Joe'");

// check for existence of field:
db.persons.find( { address : { $exists : true } } );
```

Insert & Update

- Supports bulk inserts
- Default saves are upserts
- In place updating
- Atomic transactions for single document
- Server side JavaScript execution

Insert & Update Example

```
> db.user.insert({
   first: "John",
   last : "Doe",
   age: 39
})
```

```
> db.user.update(
   {"_id" : ObjectId("51…")},
   {
    $set: {
        age: 40,
        salary: 7000}
   })
```

```
> db.user.remove({
   "first": /^J/
})
```

Indexs

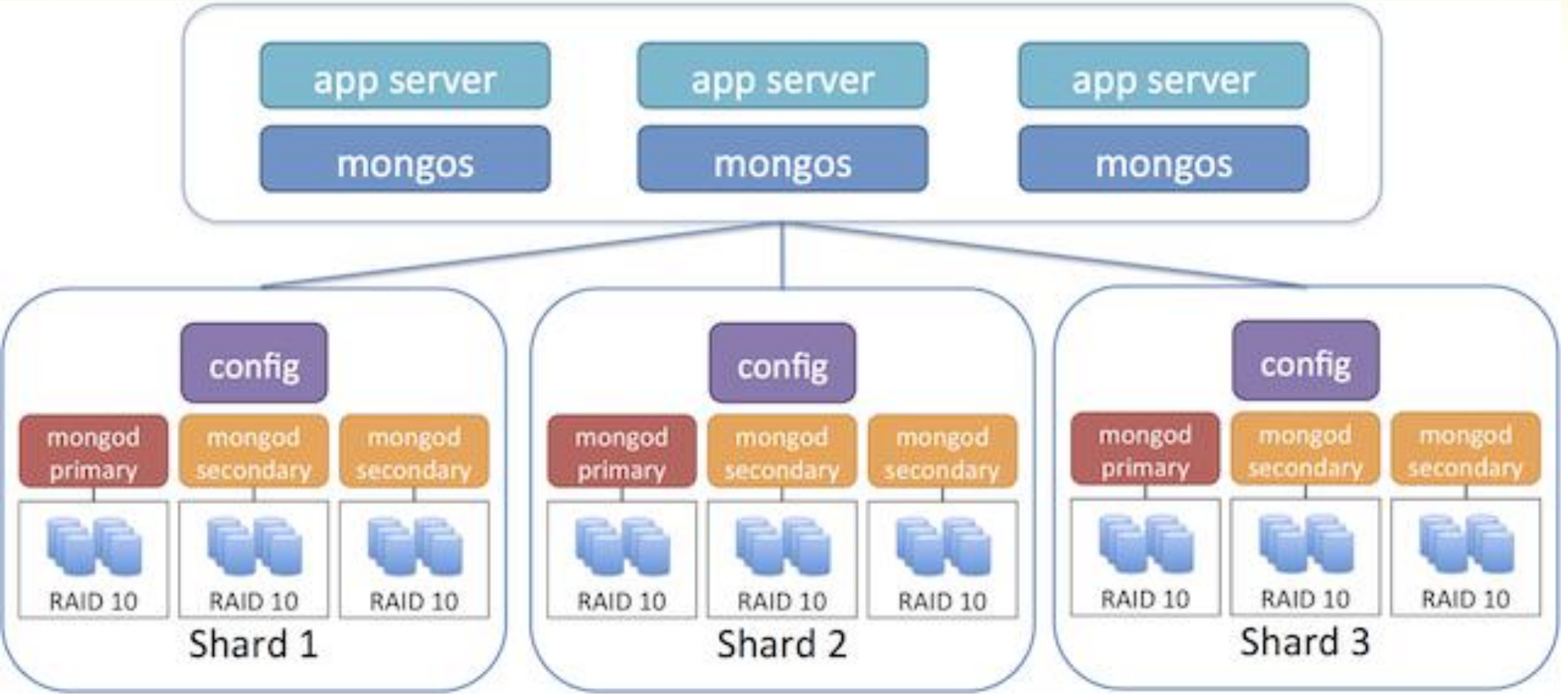
- Unique index on primary key (_*id* field)
- Create index from application code (*ensureIndex*)
- Index on embedded documents and fields
- Index on array fields (*multikey index*)
- Geospatial index

- No native full text indexing

Auto-sharding

- Partitions data across *shards*
- Any BSON document resides on only one shard
- Increases write capacity and total data size
- Data automatically distributed
- Sharding transparent to application layer
- Partitioning based on client-defined shard key
- Good shard keys are highly distributed in value and write operations
- Sharding requires config servers (minimal 3) to maintain metadata

Mongo Sharding



GridFS

- Store and retrieve files that exceed the BSON-document size limit of 16MB.
- Large blob data, limited only by storage space
- Supports many thousands of files
- Supports often changing files

Lab : Install MongoDB

- Download mongoDB from

  http://www.mongodb.org/downloads

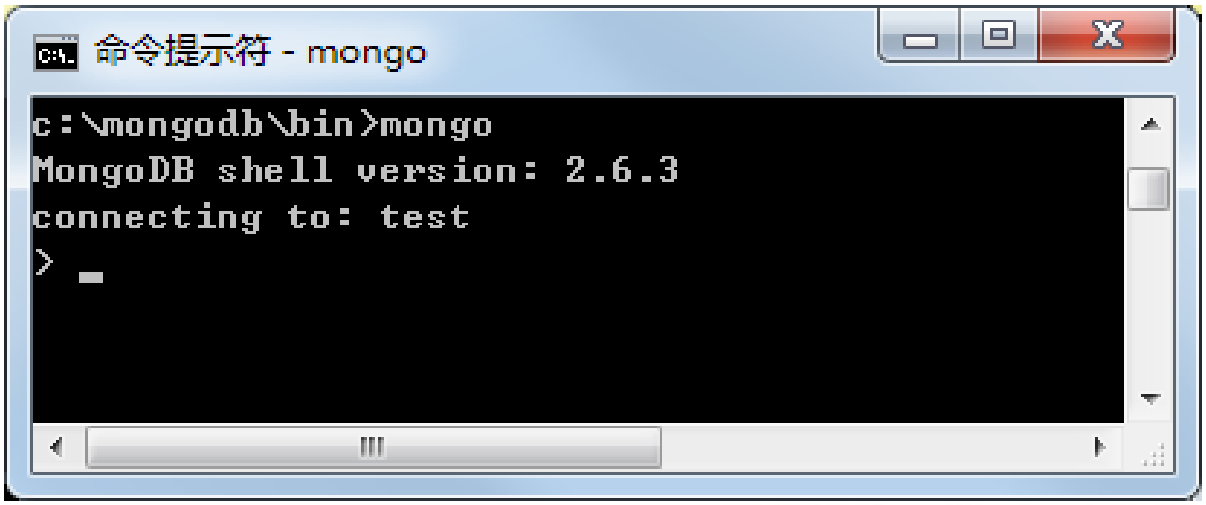
  (msi for windows)
- Create a new folder *c:\mongodb*, and install mongodb to the folder
- Create a new folder *data* inside the folder *mongodb* to store data

Lab : Start MongoDB

- Open a cmd, goto folder *c:\mongodb\bin*
- Start monfoDB by typing

```
mongod.exe --dbpath c:\mongodb\data
```

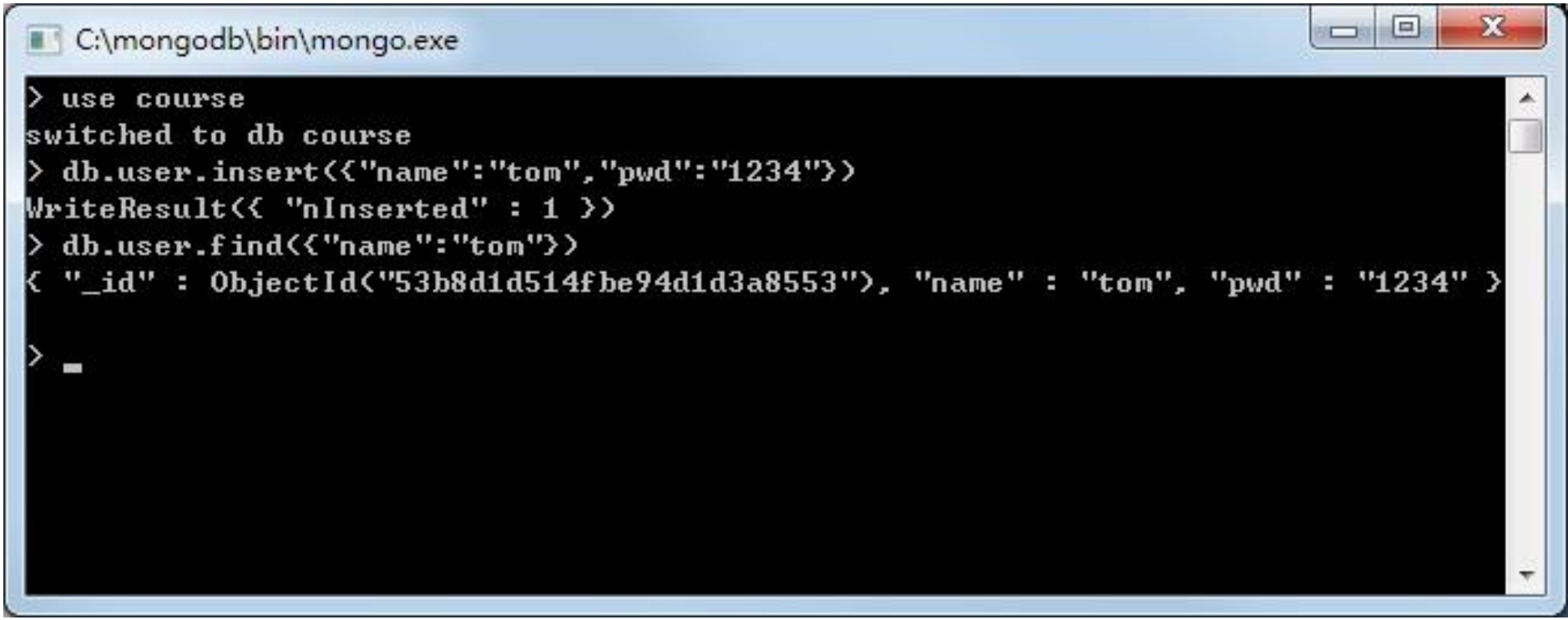
- Open another cmd, goto folder *c:\mongodb\bin* , and type `mongo.exe` to open a management window.



Lab: Add a User to MongoDB

- Go to the management window and type:

```
> use course
> db.user.insert({name:“tom",pwd:”1234”})
> db.user.find({name:”tom”})
```



Lab: Close MongoDB

- Go to the management window and type:

```
> use admin
> db.shutdownServer()
```

REST

What is REST?

- REST stands for *Representational State Transfer*.
- It describes an architecture for distributed information systems
- First described in the 2000 doctoral dissertation “Architectural Styles and the Design of Network-based Software Architectures” by Roy Fielding.
- It’s a description of how the Web works and why it works well

REST is about Architecture

- Software architecture
  - An abstraction of the elements, configurations, constraints, principles, and guidelines that govern a system's design and evolution.
- Software architectural style
  - A set of constraints that restrict a software architecture.

RESTful Examples

- Public services with RESTful APIs:
  - Twitter, Netflix, Dropbox, Flickr, Amazon S3, ...
- Products or tools with RESTful APIs
  - Glassfish Application Server Admin, Selenium WebDriver, ...
- RESTful Frameworks
  - Jersey (JAX-RS), Restlet, Restify, APEX RESTful Services, ...

Example: Amazon S3

- Amazon S3 is a Simple network Storage Service.
  - Provides cheap storage with read/write and pub/private access for the internet apps
- Support REST and SOAP APIs
  - REST API is quickly understood in 7-page tutorial
  - SOAP API WSDL is 7 pages of XML with no explanations at all!

Why REST?

- Scalable
- Human and machine usable
- Language agnostic
- Globally accessible resources
- Intuitively understandable URIs, resources, and actions.

REST Properties

- Client/server model – request/response
- Stateless: no memory of prior communications
- Cacheable
- Layered system: may use intermediary servers
- Code on demand (optional)
- Uniform interface

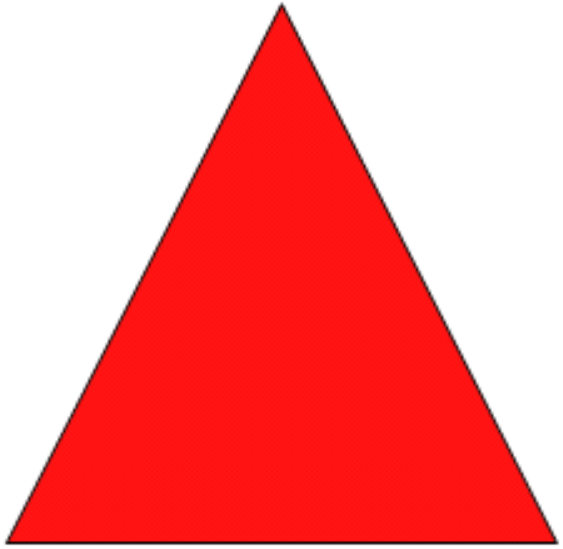
  -- Request response style operations on named resources through self descriptive representations where state changes are via hyperlinks

Uniform Interface

The REST Triangle:

- Resources
- Methods
- Representations

Resources
Nouns Unconstrained

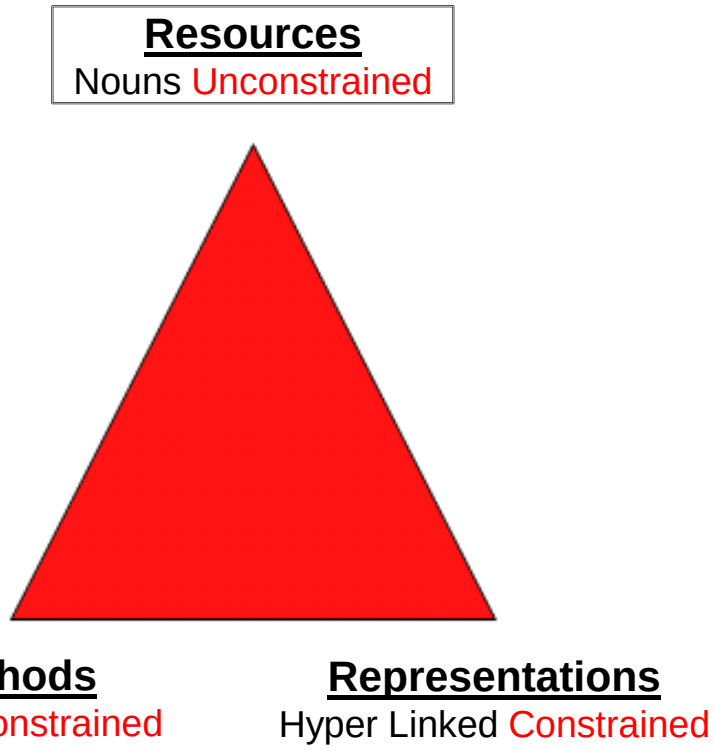


Methods
Verbs Constrained

Representations
Hyper Linked Constrained

Uniform Interfaces - Resources

- Key abstract concept
- Identified by a URI
- Distinct from underlying storage
- Semantics fixed
- Value may change over time
- Can have multiple URIs
- Can have multiple representations
- Examples:
  - http://example.org/NewOrleans/traffic/10
  - http://example.org/traffic/NewOrleans/I10
  - http://foo.com/store/orders

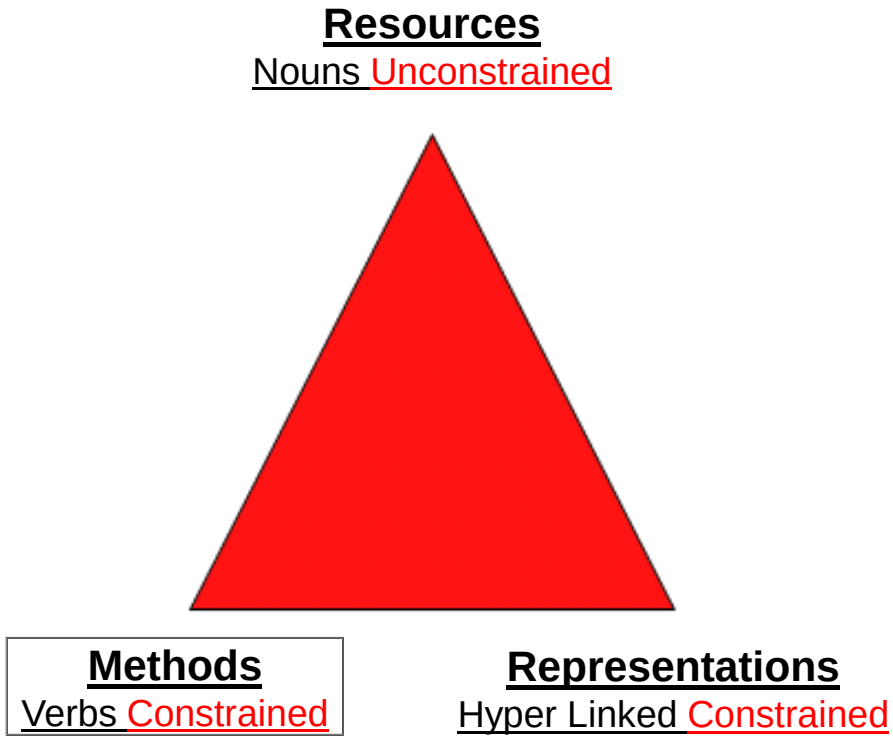


Resource Modeling - URIs

- Human readable (not necessary but it helps)
- Tends to form a hierarchy
- Use the query part appropriately
  - Use to search, filter, or possibly specify a mode
  - Identification of the resource is better in the path
    - (preferred) http://example.com/orders/100234
    - http://example.com/orders?id=100234
- Don’t make them verbs!
  - (bad) http://example.com/accounts/addaccount

Uniform Interface - Methods

- HTTP method Applying to the resource
  - GET retrieve
  - PUT update (or create)
  - DELETE delete
  - POST create sub resource
- Response codes: HTTP
  - 1xx, 2xx, 3xx, 4xx, 5xx



REST Methods

- HTTP methods POST, GET, PUT, and DELETE are often compared with Create, Read, Update, and Delete (CRUD) operations associated with database functions:

REST	CRUD	SQL
GET	Read	SELECT
POST	Create	INSERT
PUT	Update or Create	UPDATE or INSERT
DELETE	Delete	DELETE

A Example BBS System

- What message actions are required?
  - Create message
  - View message
  - View message for update
  - Delete message

RESTful Methods

- How do the message actions map in REST?

Action	HTTP Method	URL
Create Message	POST	http://bbs/messages
View Message	GET	http://bbs/messages/12
Update Message	PUT	http://bbs/messages/12
Delete Message	DELETE	http://bbs/messages/12

Design REST Methods

Method	Requirement
GET	Retrieve a resource. No modification should be done. No side effects allowed. Keep it safe.
POST	*Create or update a resource.* For non-safe, non-repeatable changes
PUT	*Update a resource.* Keep it repeatable with same results (idempotent).
DELETE	*Remove a resource.* Keep it repeatable with same results (idempotent)

Idempotence

"Idempotent operations are operations that can be applied multiple times without changing the result"

- Idempotent tasks can be retried
  - Set Salary to 60K
    - If run twice, salary will still be set to 60k
- Non-Idempotent tasks cannot safely be retried
  - Retrieve current salary and increase by 10k
    - If run twice, salary will be increased 20k, not 10k

Common Pattern for Methods

messages /

- GET - Retrieves list of all messages.
- POST - Create a new message .

messages/{mesno}/

- GET - Retrieves details for a specific message .
- PUT - Updates the specific message .
- DELETE - Deletes the message .

Are They REST ?

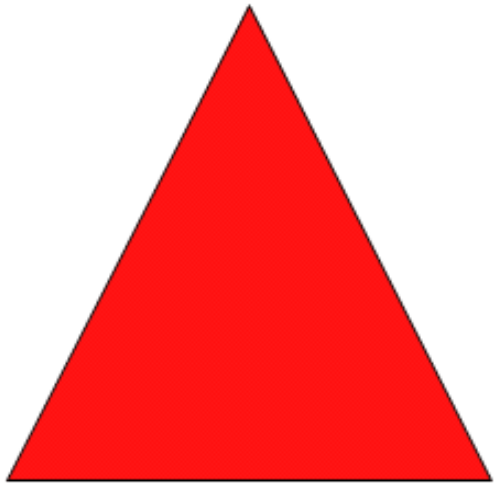
- How do these actions map to HTTP and Web pages?

Action	HTTP Method	URL
Create Message	POST	http://bbs/msg.php?m=create
View Message	GET	http://bbs/msg.php?m=view&id=12
Update Message	POST	http://bbs/msg.php?m=update&Id=12
Delete Message	GET	http://bbs/msg.php?m=delete&id=12

Uniform Interface - Representations

- Self-descriptive
- media type (Content-Type)
  - text/html
  - application/json
- Includes metadata
- Understood by all components
- May be for humans, machines or both
- Negotiated

Resources
Nouns Unconstrained



Methods
Verbs Constrained

Representations
Hyper Linked Constrained

Representation Example

Representation based on request type

- Request for XML output

```
GET /resources/forte HTTP/1.1
Host: example.com
Accept: application/xml
```

- Request for HTML output

```
GET /resources/forte HTTP/1.1
Host: example.com
Accept: text/html
```

- Could be any format you wish to support.

State Transfer Example

- Resources link together. State of the current system can be transferred along the links.

```
<resource self='http://example.com/resources/forte’>
    <name>forte</amount>
    <hostname>forte.example.com</hostname>
    <status>up</status>
    <account
       ref='http://example.com/accounts/1212’>1212</account>
    <user ref='http://example.com/profiles/jdoe’>jdoe</user>
</resource>
```

REST vs. Web Services (RPC)

	Web Services	REST
Protocols	SOAP (Simple Object Access Protocol) over HTTP.	HTTP
Request Mechanism	XML over HTTP, usually POST	HTTP
Actions (verbs)	Many different actions, which are hidden within the request body.	Standard HTTP methods (GET, PUT, POST, DELETE)
Security	Additional SOAP-specific security layer.	Web server security
Web Server	HTTP and Web server are simple conduits with much of their power and capability are bypassed	HTTP and Web server exploited to fullest extent

Example of RPC Application

- RPC Operations

```
getUser()    addUser()    removeUser() updateUser()
getLocation()addLocation()removeLocation()
updateLocation()   listUsers()  listLocations()
findLocation()     findUser()
```

- RPC Client Code

```
exampleAppObject = new ExampleApp('example.com:1234')
exampleAppObject.removeUser('001')
```

Example of REST Application

- REST Define Resources

```
http://example.com/users/
http://example.com/users/{user} (1 for each user)
http://example.com/findUserForm
http://example.com/locations/
http://example.com/locations/{location}
http://example.com/findLocationForm
```

- REST Client Code

```
userR = new Resource('http://example.com/users/001')
userR.delete()
```

Designing RESTful Services

- Key Principals
  - Everything gets a unique URI
  - Link resources together
  - Use standard methods
  - Resources have multiple representations
  - Communicate statelessly

Access RESTful Services

- REST is meant for interoperability.
- Built on same technology as the internet.
- Can be consumed in many different ways
  - Web browser
  - Web services
  - Lightweight clients
  - Command line (curl, wget)

REST Summary

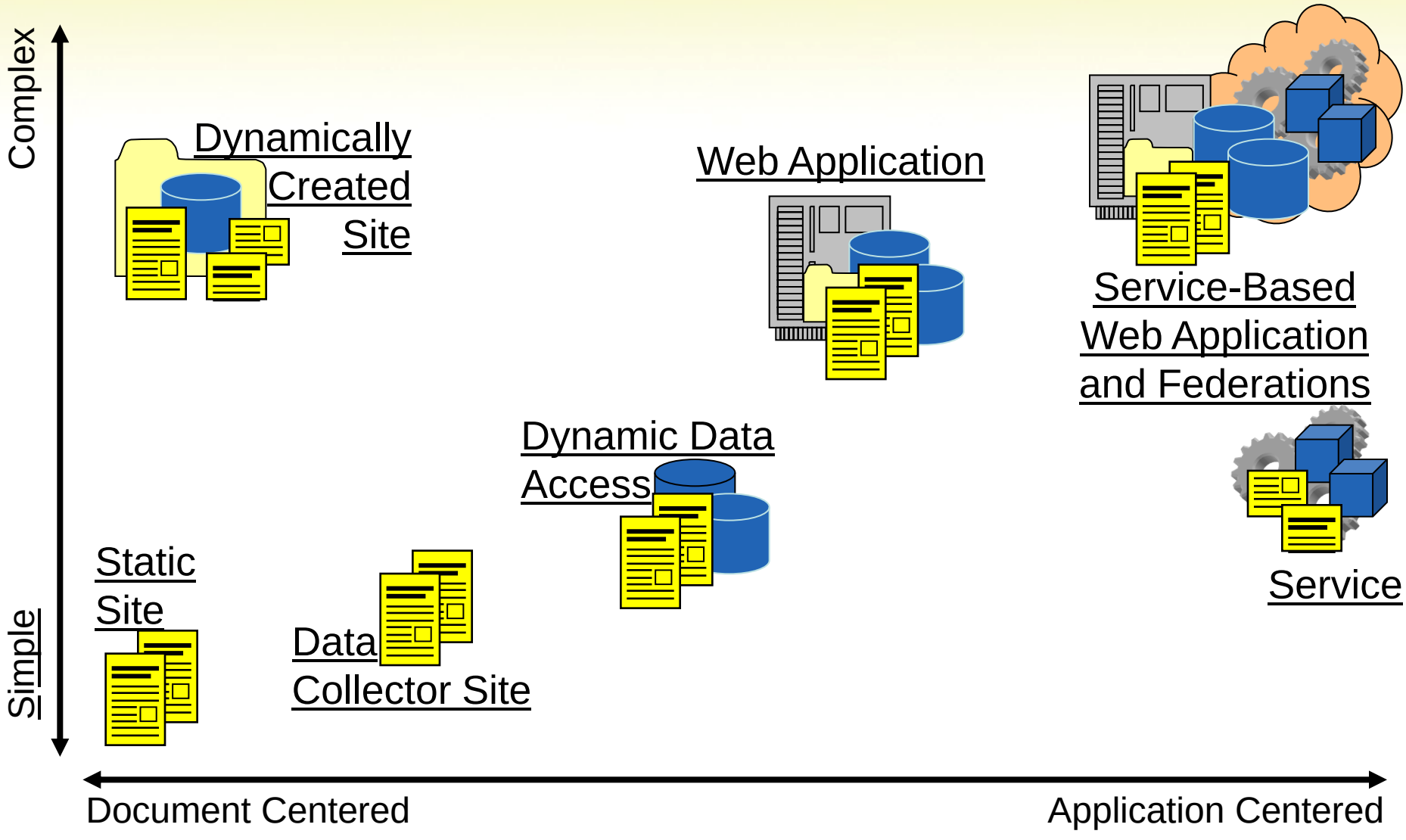
- REST is an architectural style that
  - Focuses on resources
  - Minimizes verbs
  - Leverages existing Web server capabilities
- REST encourages
  - Loose coupling
    - Separation of representation and process
    - Separation of security and logging from application
  - Maximum exploitation of Web servers and HTTP

DEVELOP WEB APPS FOR CLOUD

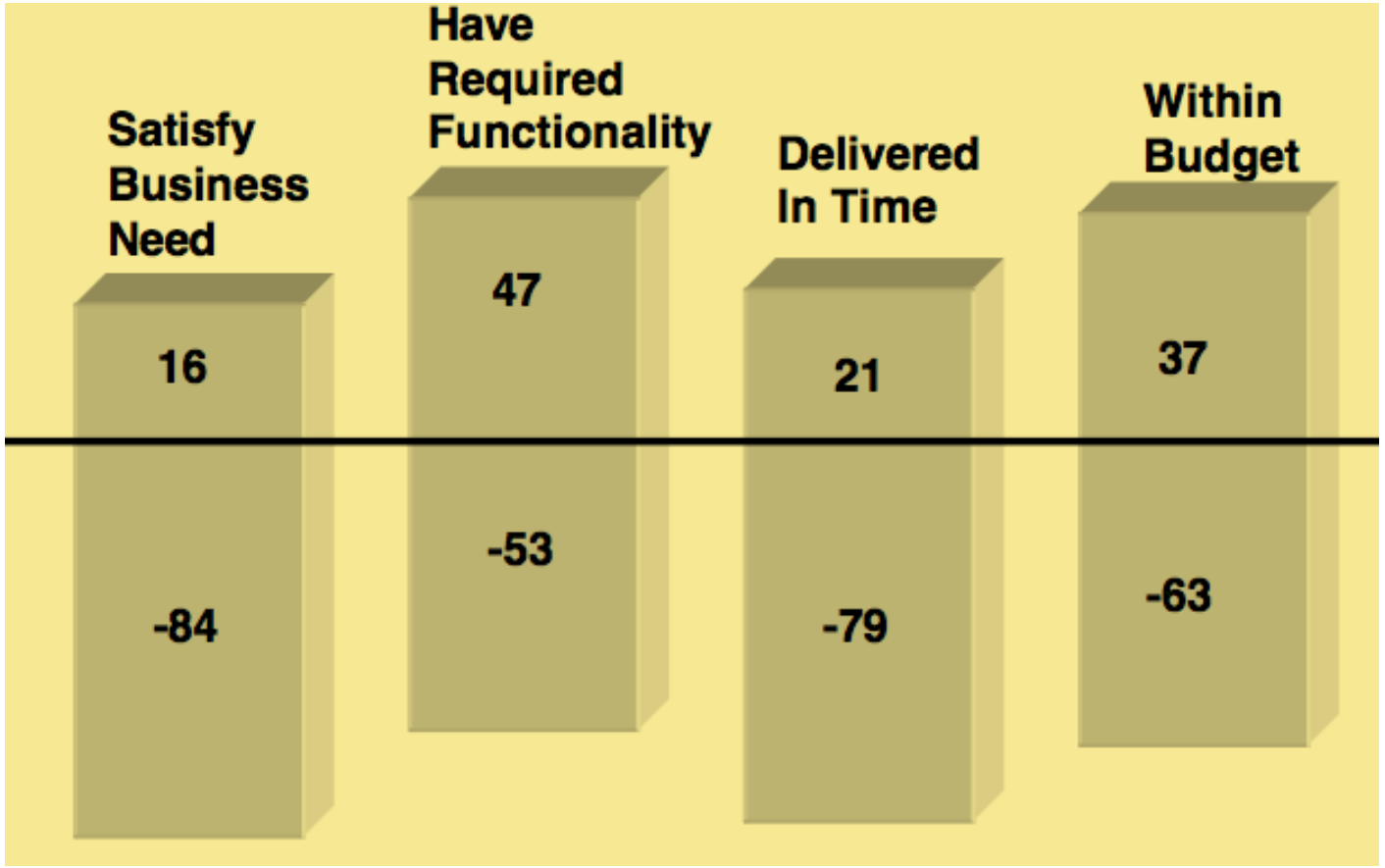
Characteristics of Web Apps

- Today's focus on large-scale and ubiquitously useable Web Applications
  - Many Users – many languages – many cultures
  - Different access mechanisms
  - Many User Agents
- Presents large volume of interrelated information (including different media) and processes
  - Appropriate presentation
  - Progression through activities – finish one thing before starting another
- Growing and increasing complexity
  - Many product iterations/versions/refinements (calls for Reuse)
  - Many developers and operators, complex handling of temporal media (e.g. publishing of company news)
  - Customization, personalization, security issues

Range of Complexity

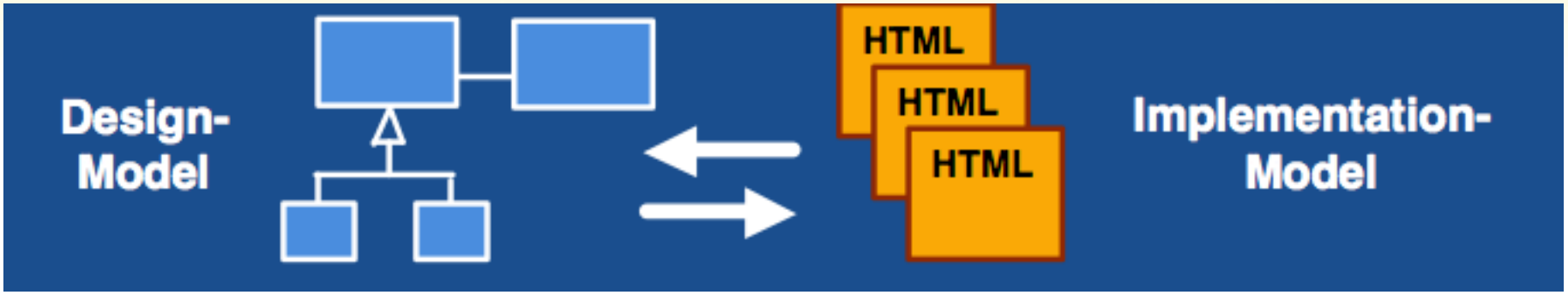


Problems with Large Web-based Projects



(Source: Epner, M., Cutter Consortium)

Web Application Development



- Still Ad-Hoc instead of a disciplined procedure
  - Lacks rigor, systematic approach
- Lack between design model and Implementation model
- Design concepts get lost in the underlying model
- Short lifecycle of a web application -> maintenance and evolution issues -> reuse issues

Web App Design

- Any complex system must begin with a cohesive fundamental design
- Multiple aspects of webapp design
  - Product Design: how the overall application will function, including:
  - Interaction (User-Interface) Design: The operation, look, and feel of the user interface
  - Software Design: The required software components and how they will interact

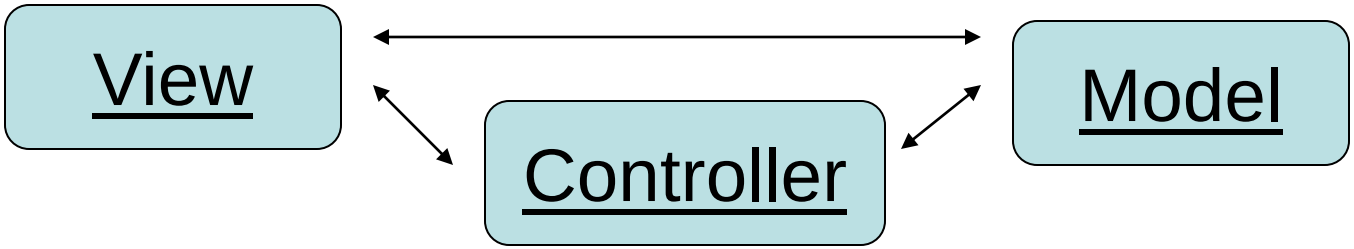
Design Patterns

- A software design pattern is an abstraction of some commonly used software design
- This abstraction includes enough information to enable reuse of the pattern with new applications

- A pattern for architectural (high-level) design is also called an architectural style

Model-View-Controller

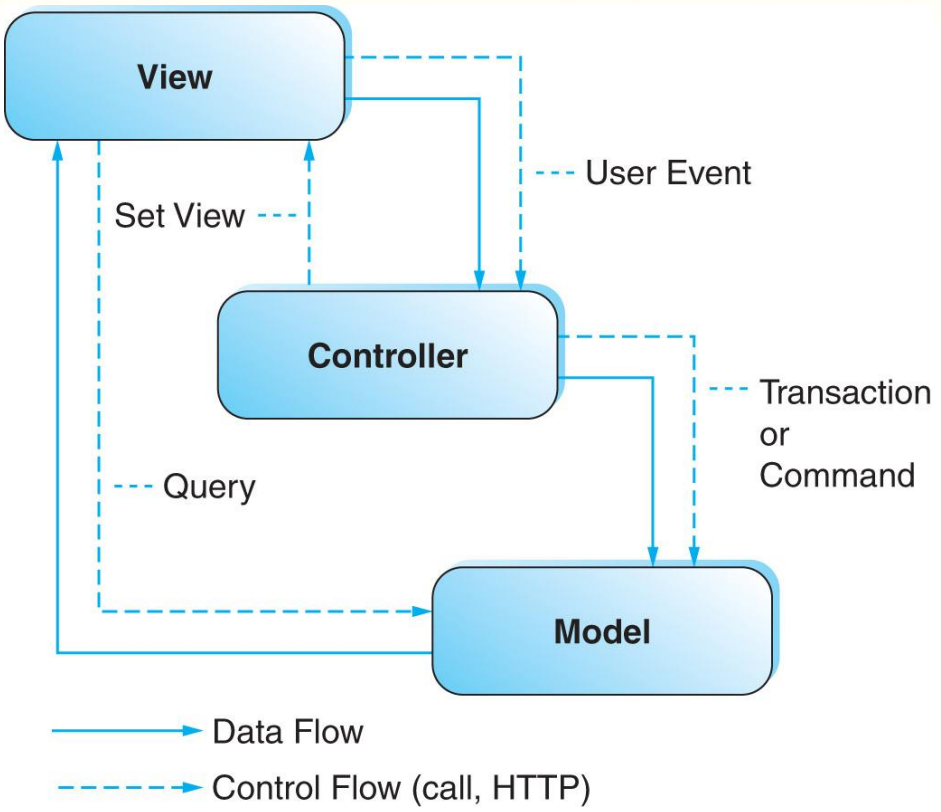
- Many webapps are based on the well-known model-view-controller design pattern



- The MVC pattern specifies the responsibilities of three major components, and the nature of their interactions

MVC Component Responsibilities

- View: Present the user interface
- Model: Maintain the application state
- Controller: Handle user actions



Model Role

- The Model maintains the application state, which includes:
  - persistent information stored in databases
  - current information related to active sessions
- Responsibilities include:
  - applying rules / transactions to modify state
  - providing information to the View as required
  - taking instructions from the Controller

View Role

- The View presents a user interface, including information and transactional controls

- Responsibilities include:
  - Present required information to user
  - Request data from Model as needed (to create displays for the user)

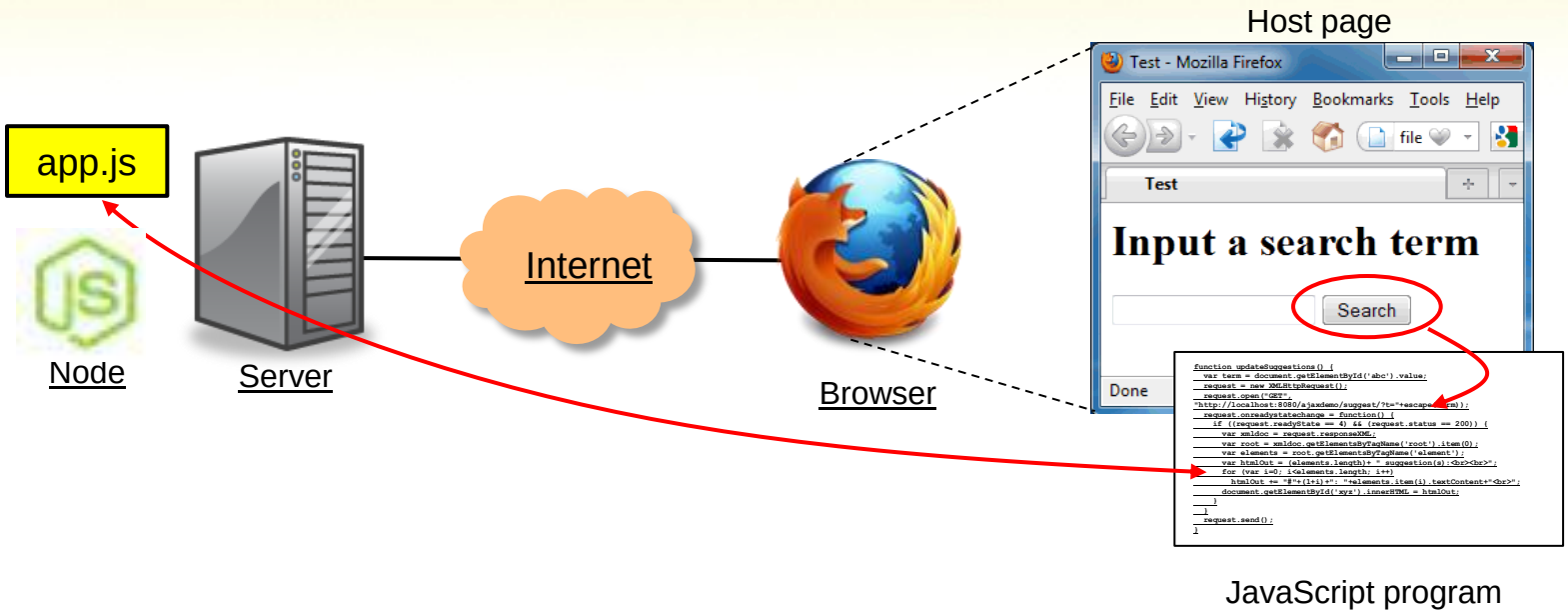
Controller Role

- The Controller handles user actions
- Responsibilities include:
  - Handle user actions
  - Validate user requests (correctness, completeness)
  - Invoke Model components to handle requested transactions
  - Set the appropriate next View perspective

Single-Page Web App

- One host (main) HTML file
- One screen at any given time
- Generating (or toggling) other screens dynamically, using JS
- Instead of navigating between different pages, one navigates between different sections of one web page
- Maintaining state – what's currently in the DOM
- Maintaining data – what's needed for generating DOM

Building Web Applications

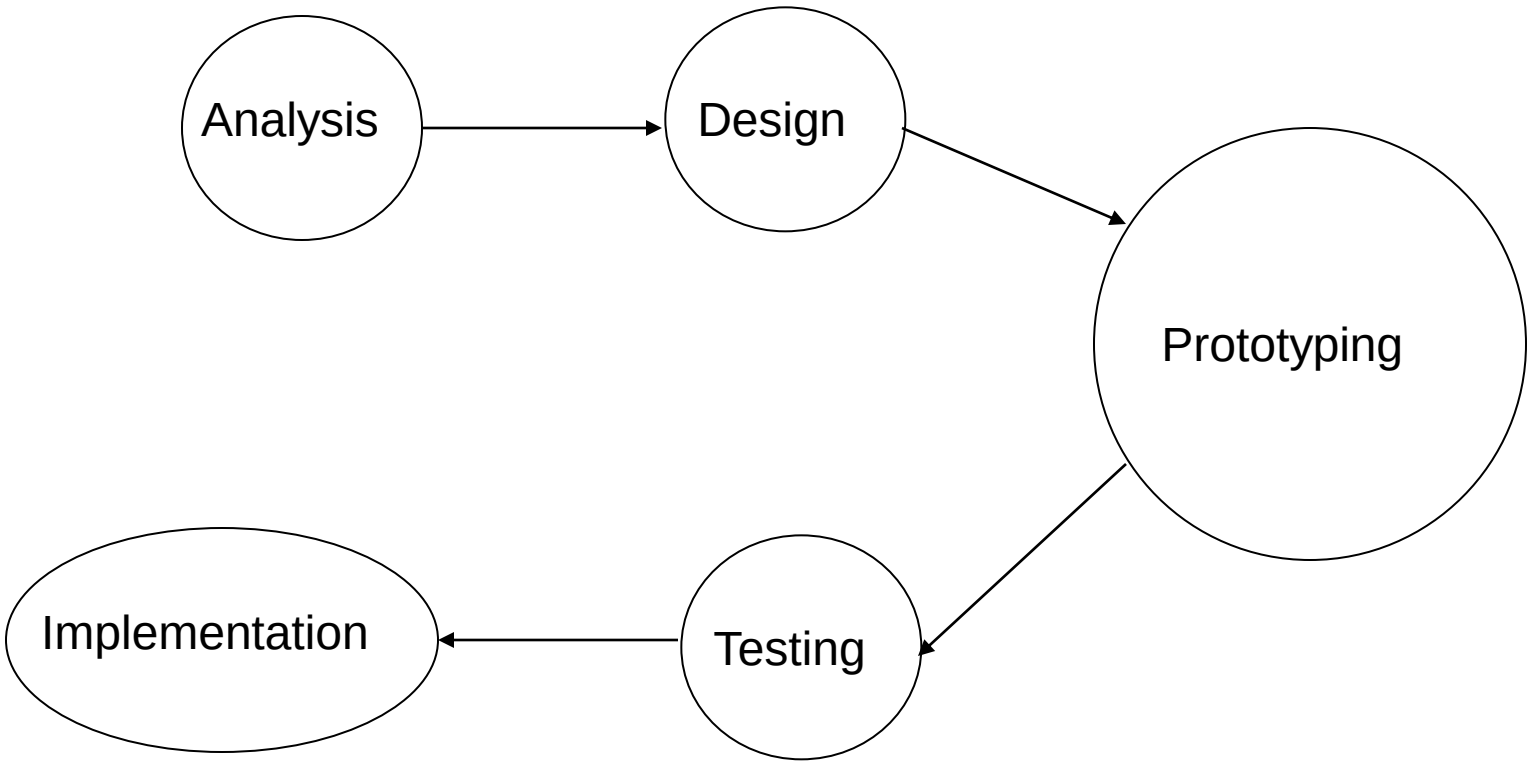


- Host page: A HTML page that we'd like to make interactive.
- CSS is used to decorate the main page.

Building Web Applications (2)

- Client-side script: A JavaScript program that
  - registers handlers for relevant DOM events, such as inputs or mouse clicks
  - requests additional data from the server using Ajax;
  - integrates the responses with the web page using the DOM
  - Send back to the server the user's data using Ajax
- Server side: A program that supplies and receives the data
- Client-side script runs in browser, server-side using Node.js

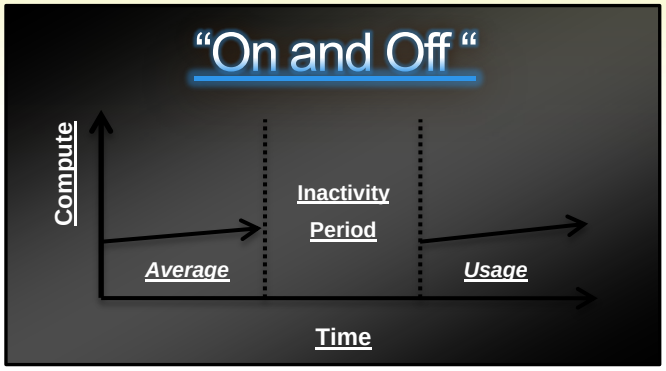
Web Development Cycles



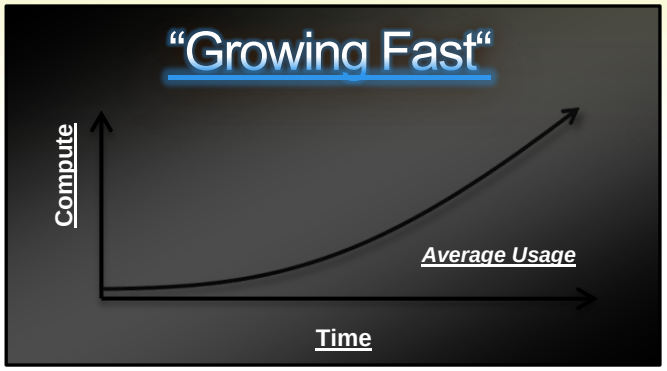
Security Requirements

- Privacy - All user information are protected
- Authentication/Access Control- Only authorized users are allowed to access the resources
- Integrity - User and application data cannot be tempered with
- Auditing - Keeping audit logs and audit trails and ensuring their integrity

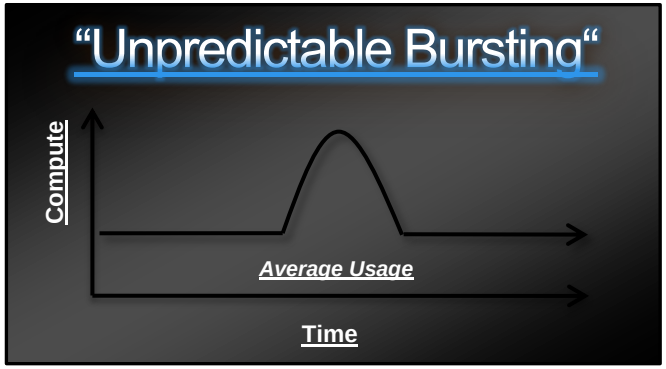
Applications Fit for Cloud



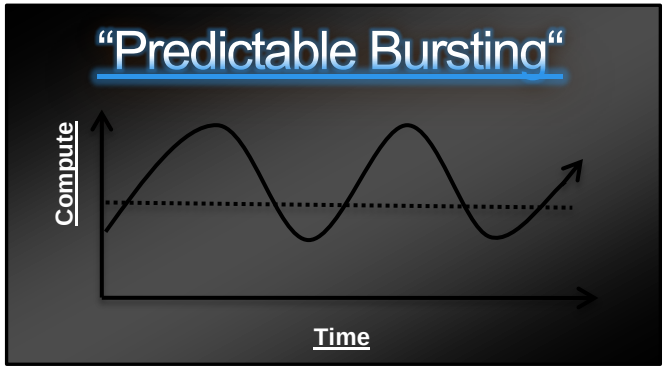
- On & off workloads (e.g. batch job)
- Over provisioned capacity is wasted
- Time to market can be cumbersome



- Successful services needs to grow/scale
- Keeping up w/ growth is big IT challenge
- Cannot provision hardware fast enough



- Unexpected/unplanned peak in demand
- Sudden spike impacts performance
- Can't over provision for extreme cases



- Services with micro seasonality trends
- Peaks due to periodic increased demand
- IT complexity and wasted capacity

http://cloudyinnewyork.com/2012/10/26/to-cloud-or-not-to-cloud/

Cloud Apps Development

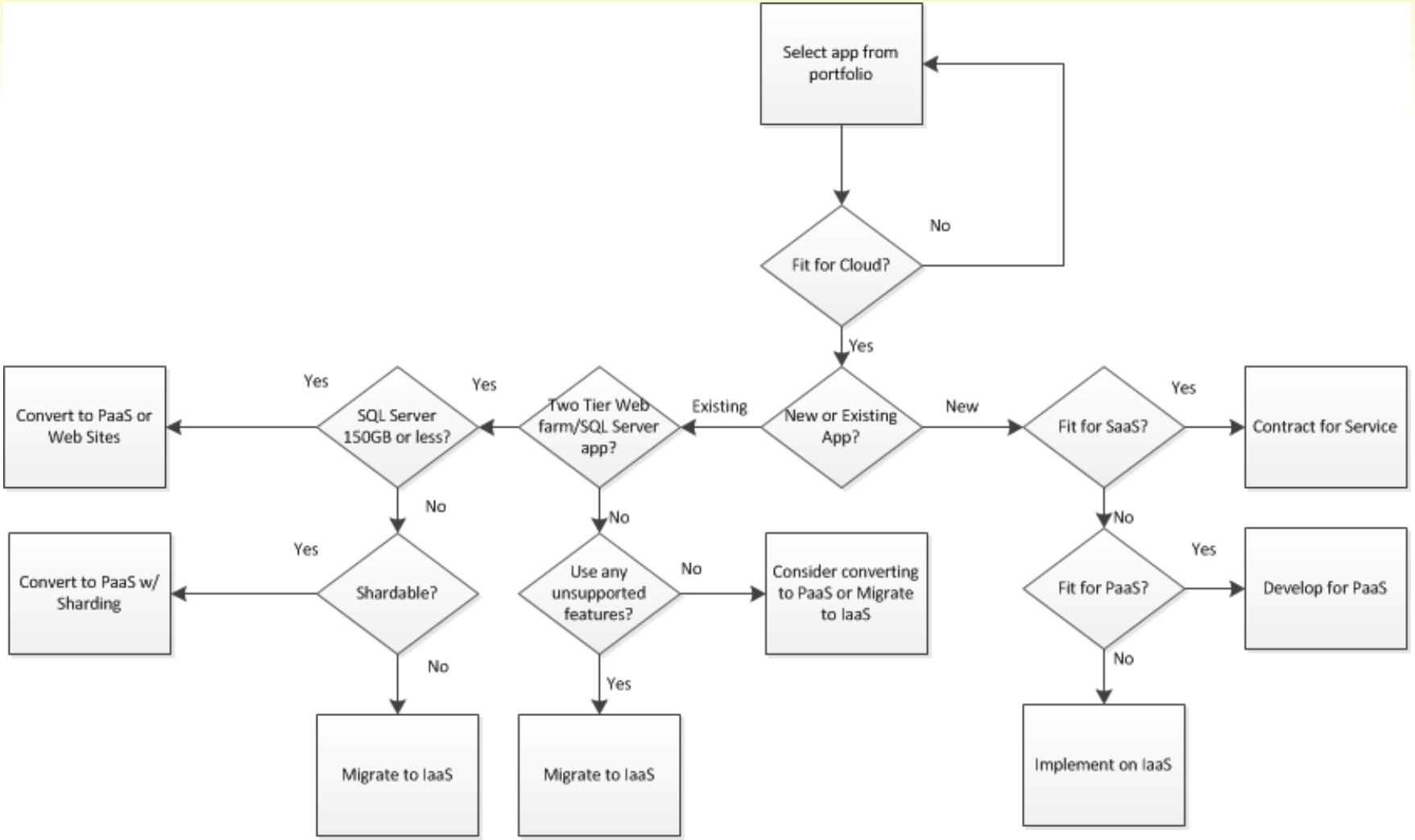
- Cloud software development
  - Design and develop an application for the cloud
  - Especially for the public PaaS cloud platforms

- Typical steps in cloud software development
  - Choose a development stack of technologies
  - Choose a cloud platform + services
  - Design the application for the cloud
  - Develop the application using the cloud APIs
  - Deploy and run the application in the cloud

Transition to Cloud Development

- Transition to cloud development
  - New architecture (based on SOA)
  - New programming paradigms
    - E.g. NoSQL databases
  - New APIs
    - E.g. Amazon S3
  - New deployment model
    - Git + vendor-specific continuous integration process

Moving an App to Cloud

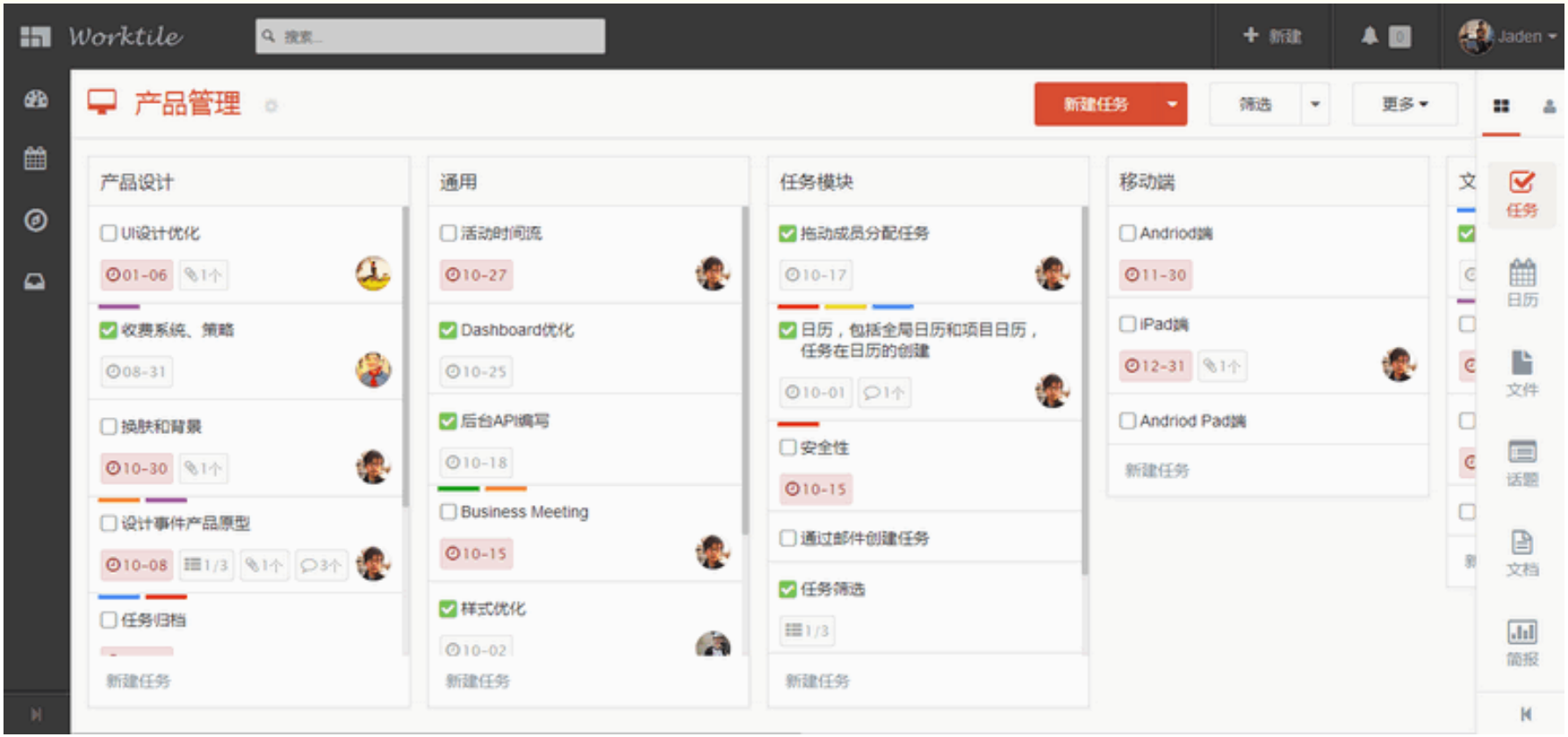


See CloudyInNewYork.com

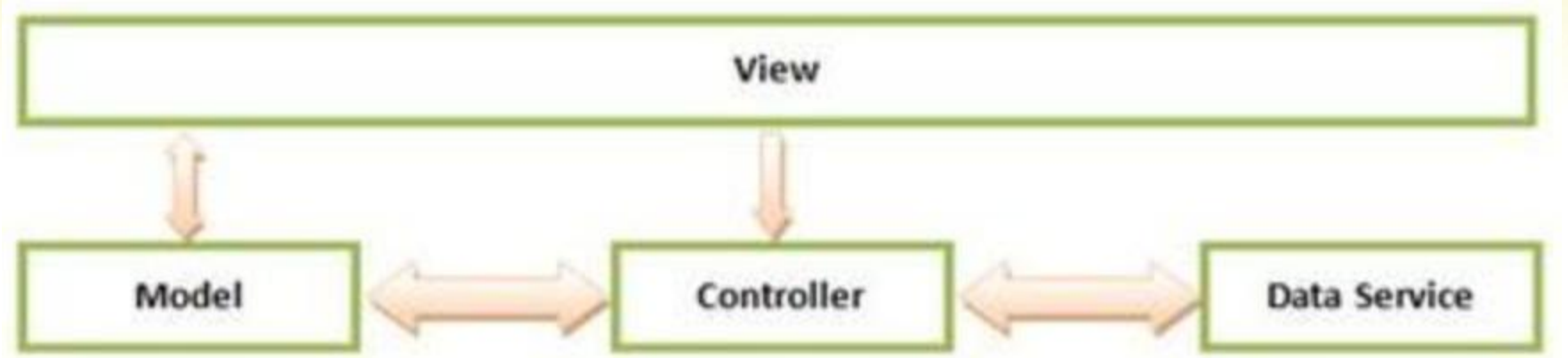
Example: WorkTitle

- A team cooperation tool
- Web-based cross-platforms, visit from browsers
- Single page application. Interface and feeling of native apps, such as drag and drop
- Instant message, every modification in one client will be automatically refreshed in other clients.
- Provide stable service
- Already more than 100 thousands users

SPA of WorkTitle

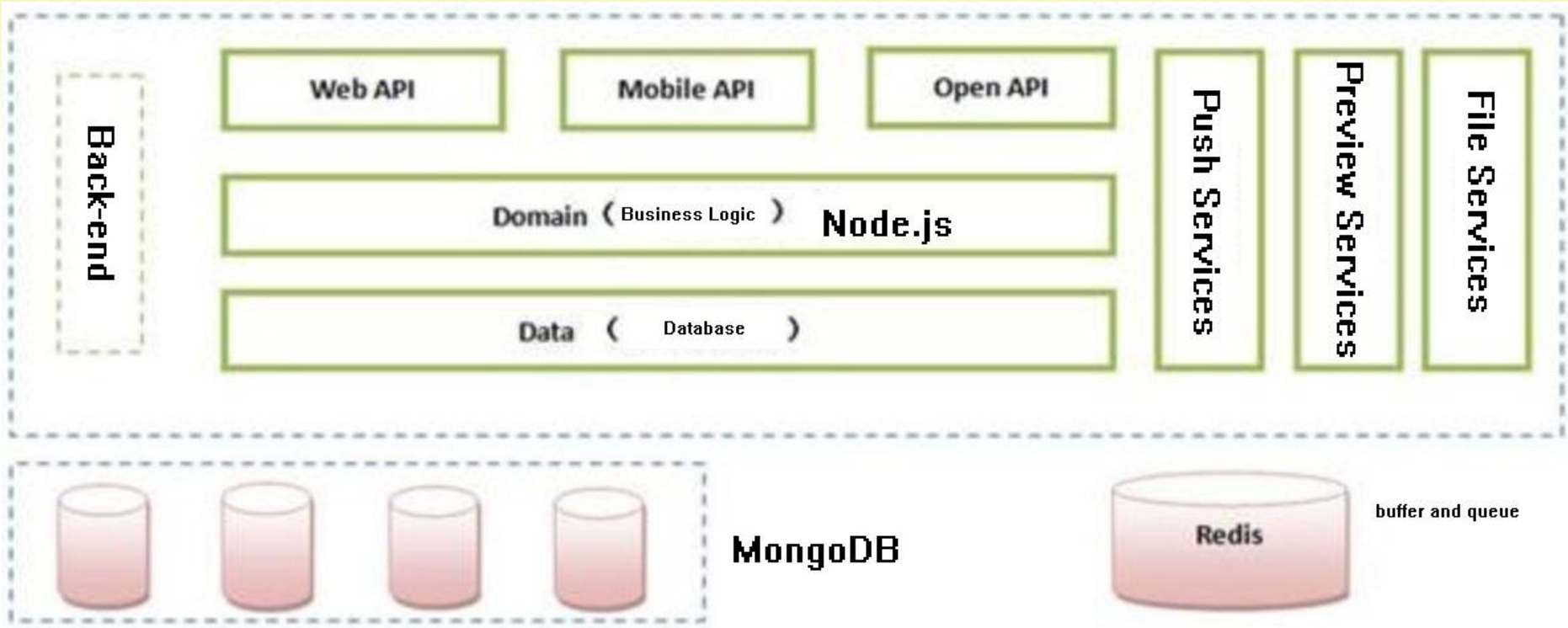


Front-side: AngularJs



- Two-way data binding
- Declarative semantic tags
- Modules
- Dependency injection

Back-End of WorkTitle



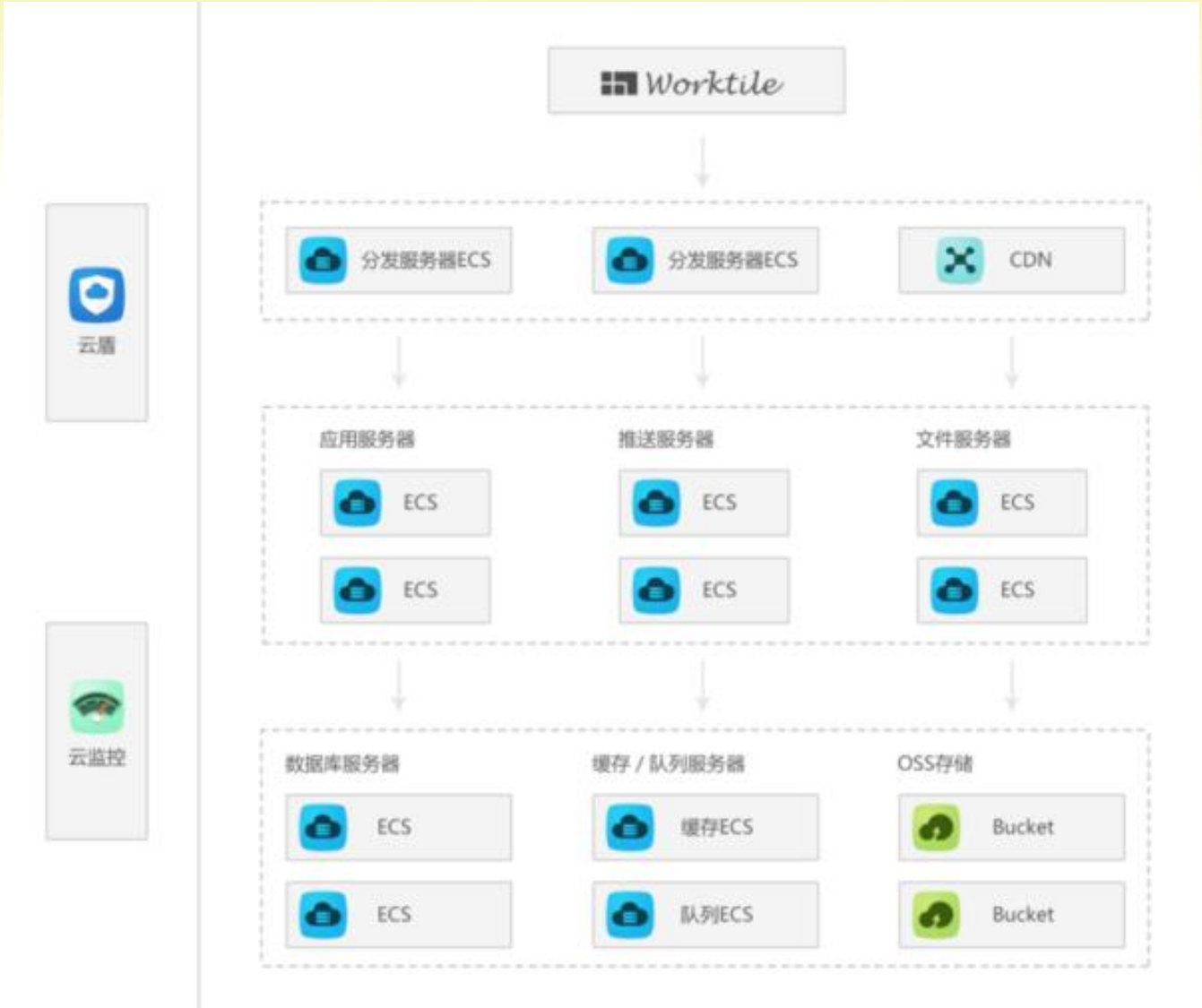
- Preview service: Microsoft Office Web App
- Push service: ejabberd (core technology)

Long Polling vs Short Polling vs Websocket

- Short polling : uses a timer and asks the server if new data is available
- Long Polling: uses an event based algorithm, more realtime.

- Extensible Messaging and Presence Protocol (**XMPP**) is a near-real-time communications protocol for message exchange.
- **ejabberd** is the world's most popular XMPP server. It can

WorkTitle on Aliyun



Cloud Architecture Techniques

OpenStack/Docker

What is Virtualization?

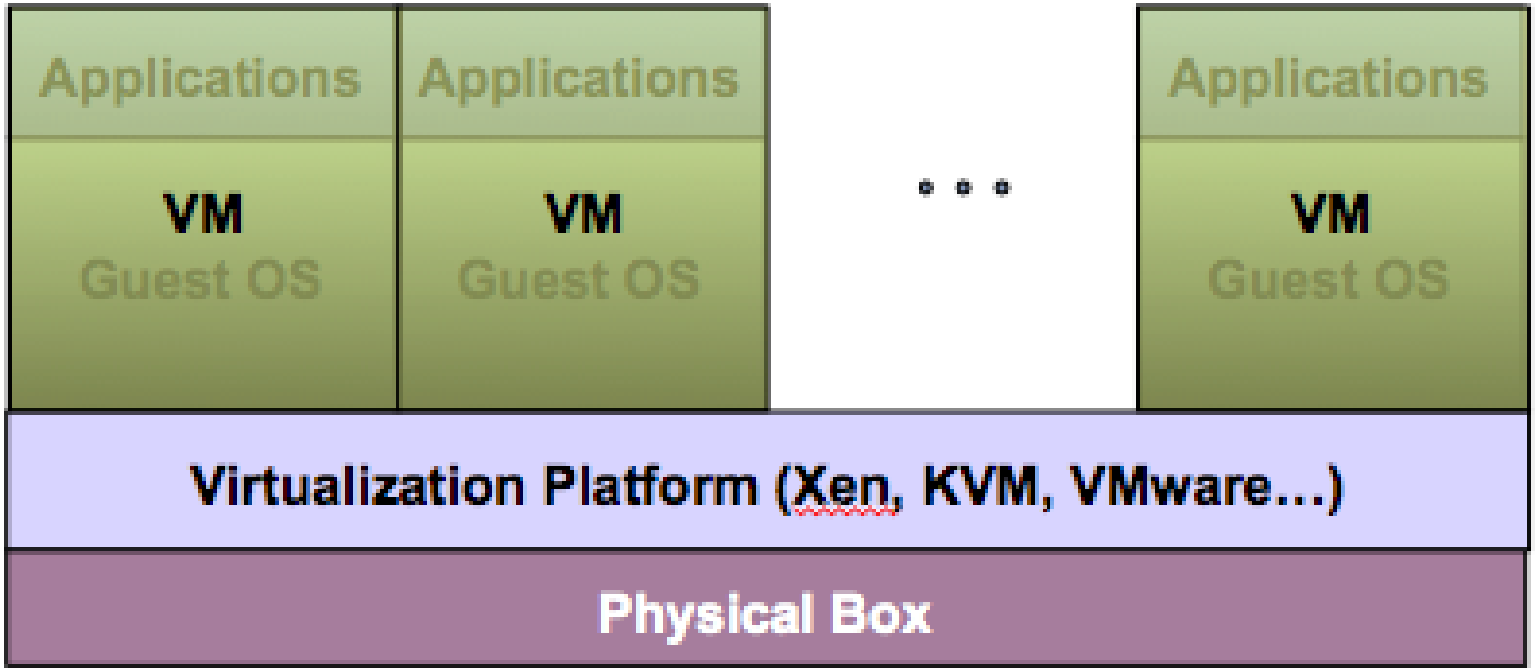
- Virtualization
  - Running several virtual machines (virtual computers) inside a single physical machine
    - Supported by special software called hypervisor
  - Uses resources more efficiently
    - E.g. 12 GB physical RAM is shared to 6 virtual machines with 4 GB shared RAM each
    - Most applications use 1-5% of the CPU
      - A single shared CPU can serve thousands of users
  - Reduces costs due to better utilization

Virtualization Definitions

- Virtualization is the ability to run multiple operating systems on a single physical system and share the underlying hardware resources
- Virtualization is used to improve IT throughput and costs by using physical resources as a pool from which virtual resources can be allocated.
- Virtual machine (VM) software is a program that emulates a physical machine
- A VM needs to act *exactly* like its physical machine
- A VM instance is simply a file that represents an actual machine and its state

Virtualization Architecture

- A Virtual machine (VM) is an isolated runtime environment (guest OS and applications)
- Multiple virtual systems (VMs) can run on a single physical system



OpenStack & Docker

Open Frameworks

PaaS
Application Centric

IaaS
Resource
Centric

OPENSTACK

OpenStack

- An open source cloud computing framework
- It is primarily used to enable **infrastructure as a service**
- OpenStack began in 2010 as a joint project of Rackspace Hosting and NASA. Currently, it is managed by the OpenStack Foundation

OpenStack Mission

To produce the

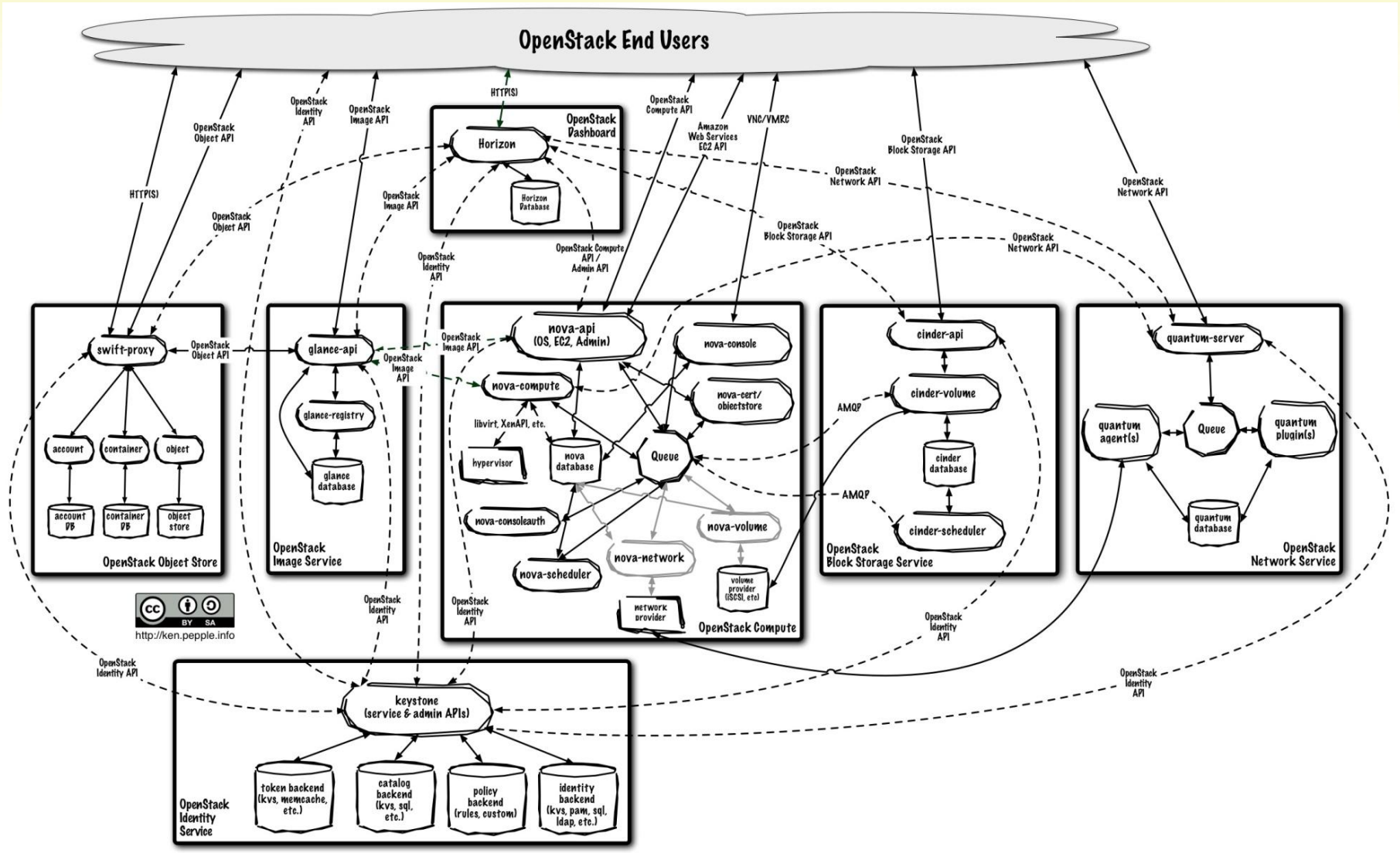
ubiquitous open source cloud computing platform

that will meet the needs of public and private clouds regardless of size, by being simple to implement and massively scalable.

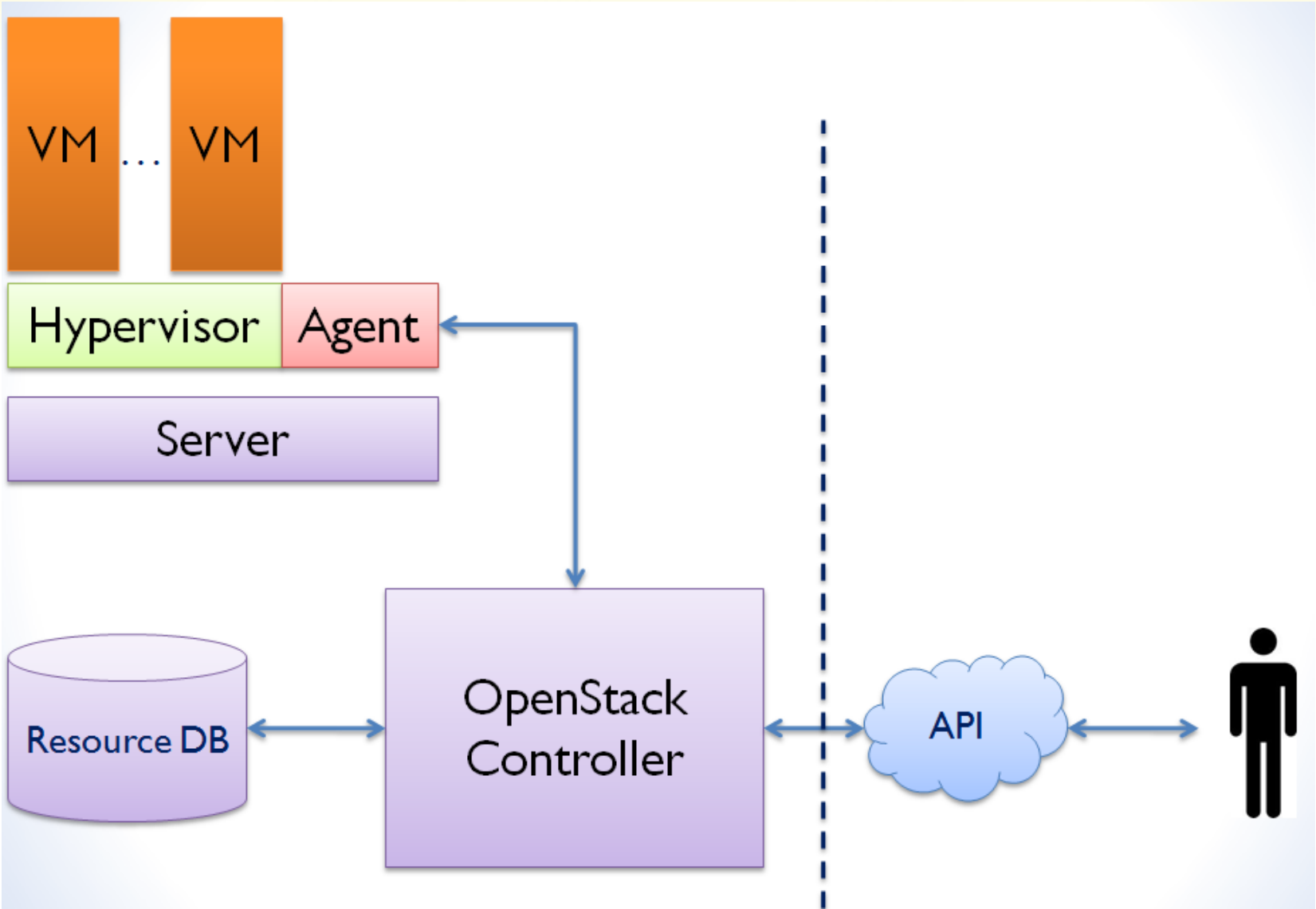
Trends and Themes of OpenStack

- OpenStack Powered Planet – delivering on cloud interoperability and federation across vendors and geographies
- Applications on OpenStack – new community app catalog and first app guide provide more support for developers targeting OpenStack clouds
- Platform for the Next 10 Years – OpenStack gives users the ability to integrate and test new technologies like containers as they emerge

OpenStack is Complicated



But, Basic of OpenStack



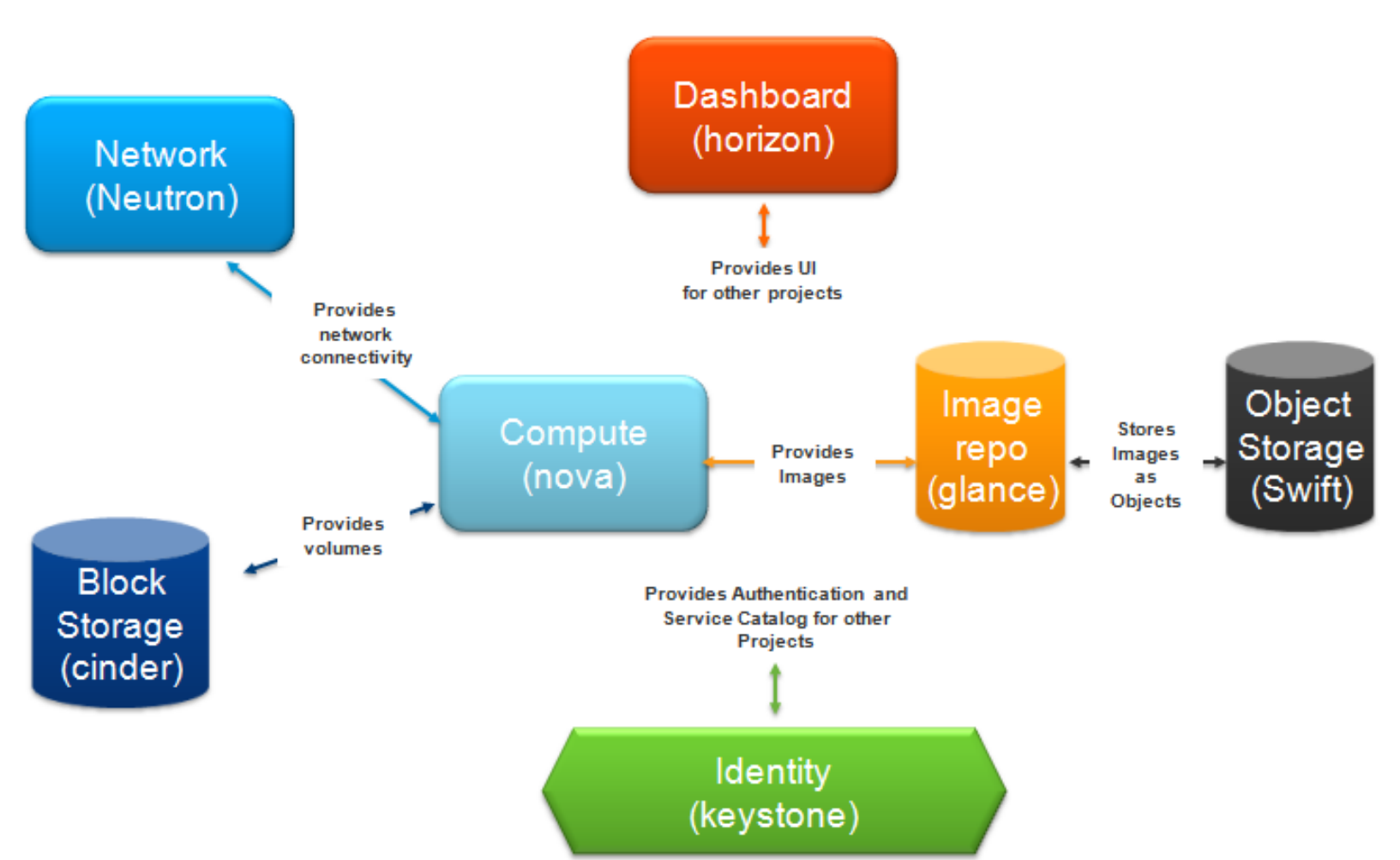
VMs as OpenStack Cloud Resources

- VMs run on top of hypervisors, and Agent process runs on physical machine beside hypervisor.
- User requests VM (resource) from controller, controller finds available resources in pool, contacts appropriate Agent
- Agent interprets command, instructs hypervisor to boot VM with user provided OS image
- Openstack returns handle to the user

Components of OpenStack

- Compute (Nova)
- Object Storage (Swift)
- Block Storage (Cinder)
- Networking (Neutron)
- Dashboard (Horizon)
- Identity Service (Keystone)
- Telemetry (Ceilometer)
- Orchestration (Heat)
- Database (Trove)
- Bare Metal Provisioning (Ironic)
- Multiple Tenant Cloud Messaging (Zaqar)
- Elastic Map Reduce (Sahara)

OpenStack Main Components



Compute (Nova)

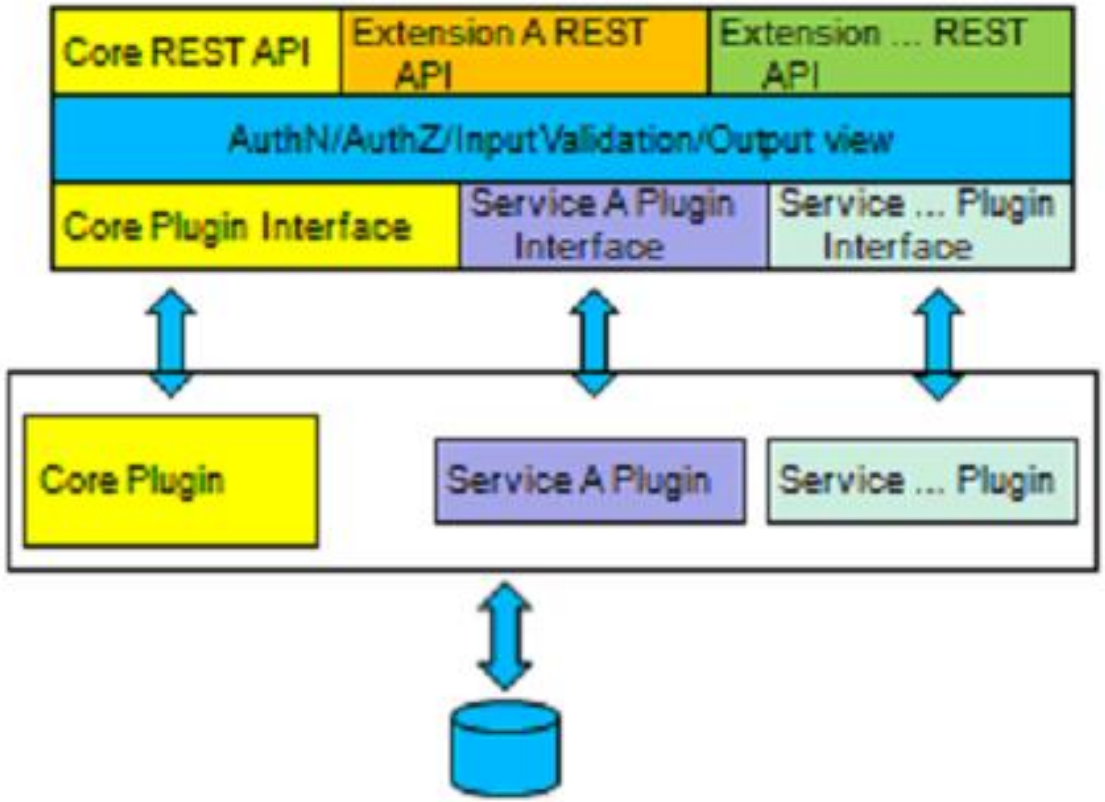
- Provision and manages virtual machines. It is a major part of an IaaS system.
  - **Nova-api service**: Accepts and responds to end user compute API calls.
  - **Nova-compute service**: A worker daemon that creates and terminates virtual machine instances through hypervisor APIs.
  - **Nova-schedule**r: Takes a virtual machine instance request from the queue and determines on which compute server host it runs.
  - **Rabbit MQ Server**: A central hub for passing messages between daemons

Storage (Swift)

- Object Storage:
  - –Object Storage provides a fully distributed, API-accessible storage platform that can be integrated directly into applications or used for backup, archiving and data retention.
- Block Storage:
  - –Block Storage allows block of storages to be exposed and connected to compute instances for expanded storage, better performance and integration with enterprise storage platforms

Networking(Neuron)

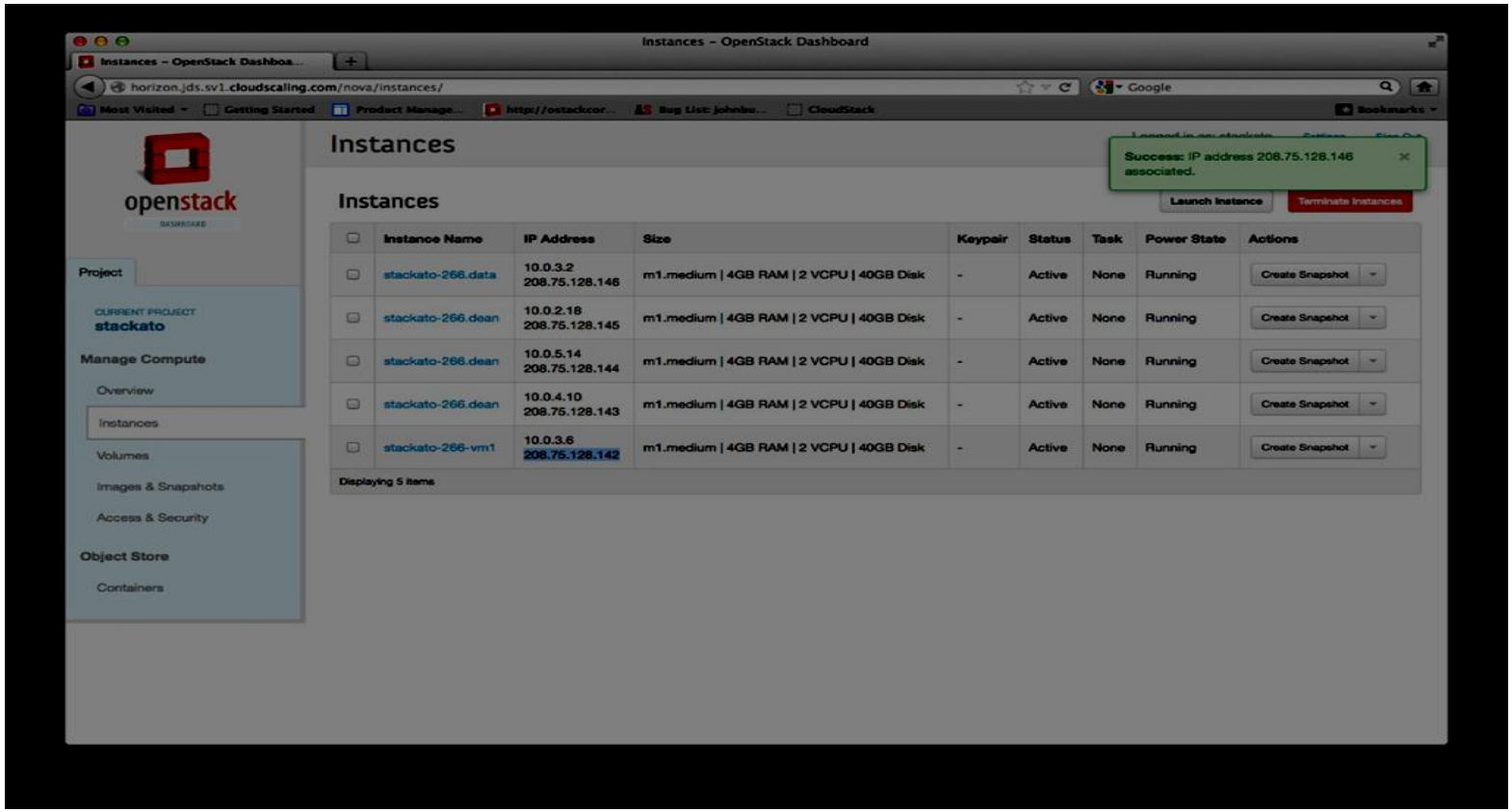
- Neutron is an OpenStack project to provide "networking as a service" between interface devices.
- It accepts and routes API requests to the appropriate OpenStack Networking plug-in devices for action.



Dashboard (Horizon)

- Dashboard provides a web based user interface to OpenStack services including Nova, Swift, Keystone, etc.
- The OpenStack dashboard provides administrators and users a graphical interface to access, provision and automate cloud-based resources.

Sample Dashboard



Identity Service (Keystone)

- Keystone is the identity service used by OpenStack for user authentication.
- It performs the following functions:
  - Tracking users and their permissions.
  - Providing a catalog of available services with their API endpoints.

Keystone and Dashboard

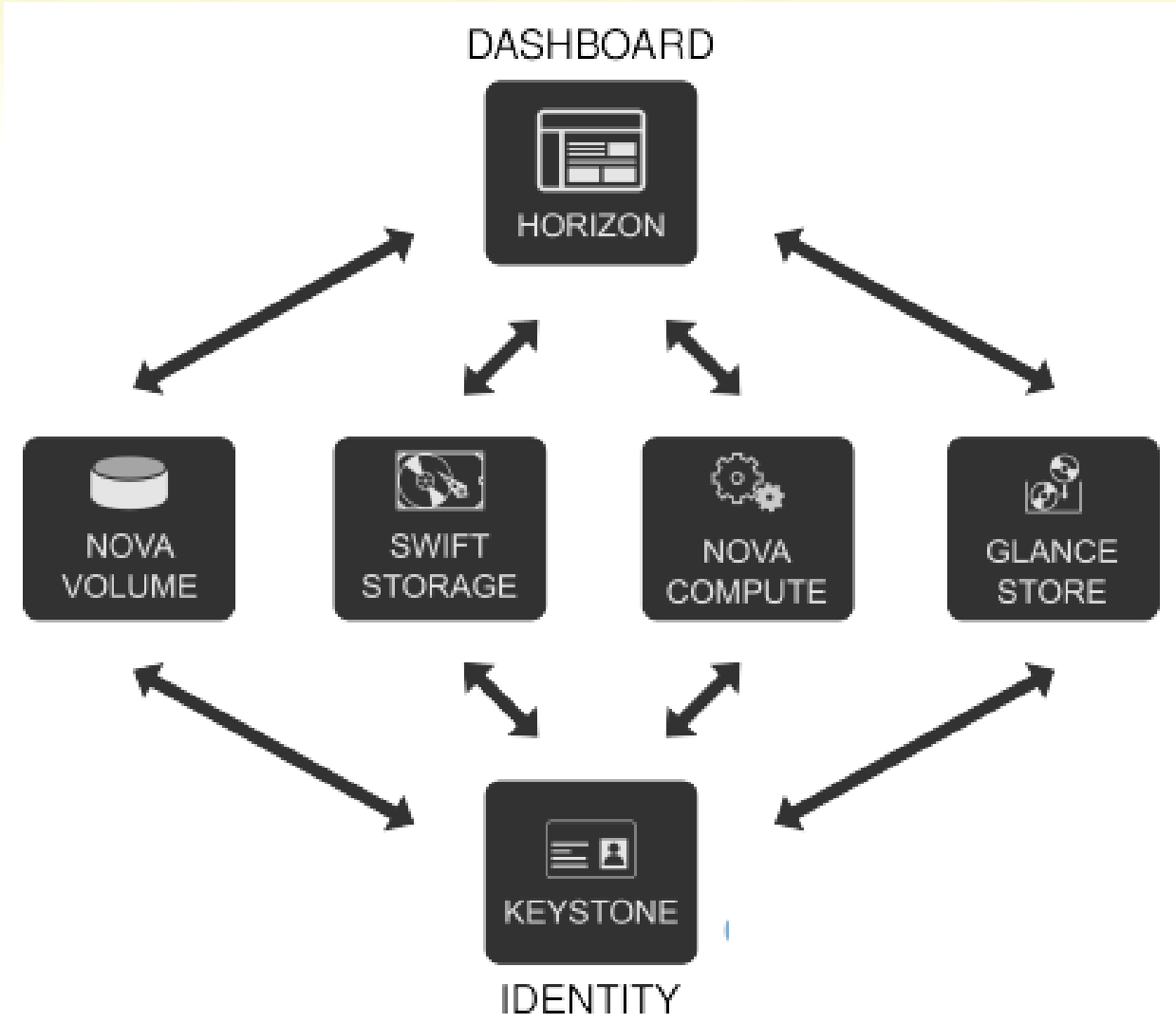


Image Service (Glance)

- The OpenStack Image Service provides discovery, registration and delivery services for disk and server images.
  - glance-api: Accepts Image API calls for image discovery, retrieval, and storage.
  - glance-registry: Stores, processes, and retrieves metadata about images.
  - Storage repository for image files: Various repository types are supported.

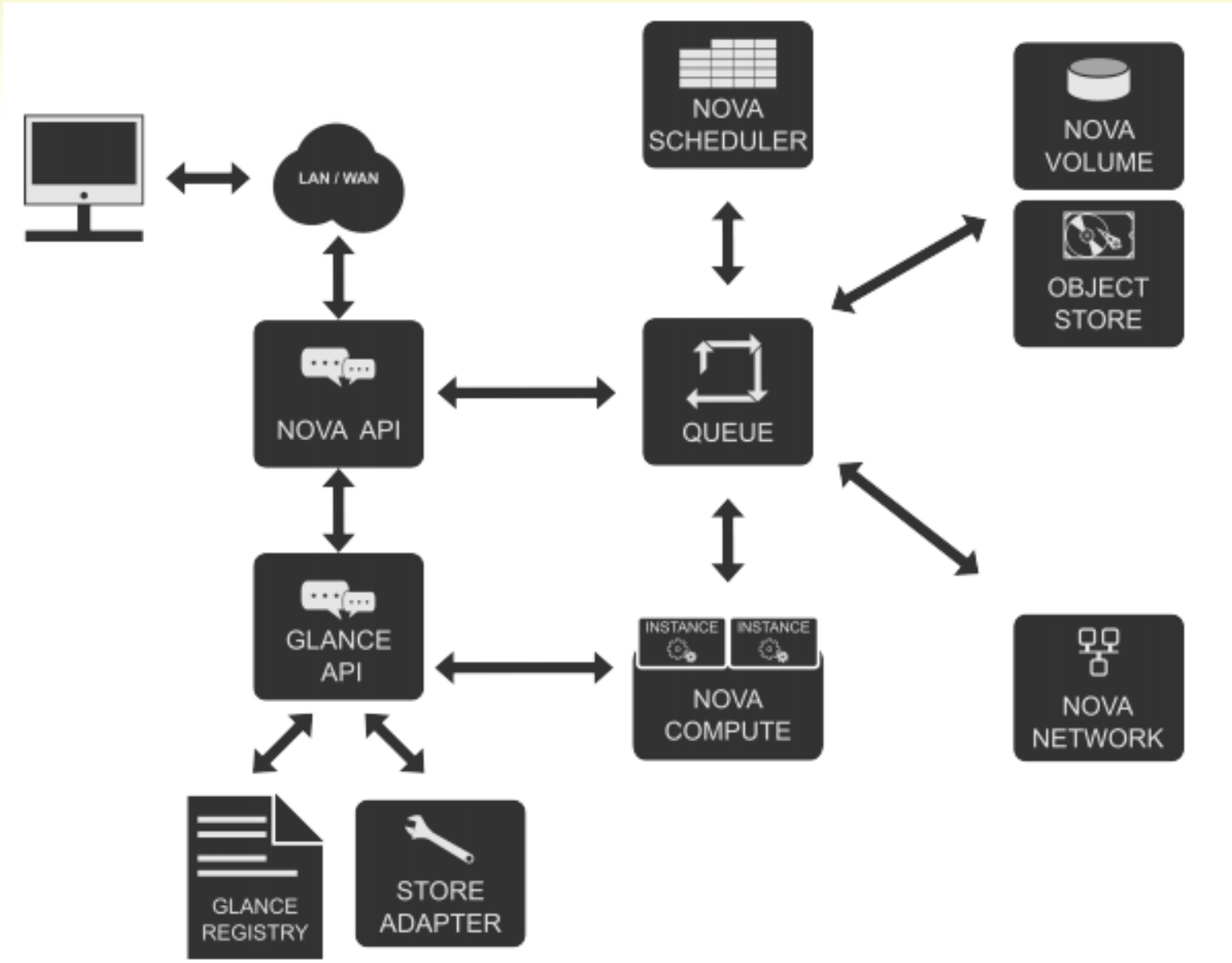
Telemetry (Ceilometer)

- The OpenStack Telemetry service aggregates usage and performance data across the services deployed in an OpenStack cloud.

Orchestration (Heat)

- OpenStack Heat program is to create a human- and machine-accessible service for managing the entire lifecycle of infrastructure and applications within OpenStack clouds.

Architecture of OpenStack



nces

Use Cases of Openstack:

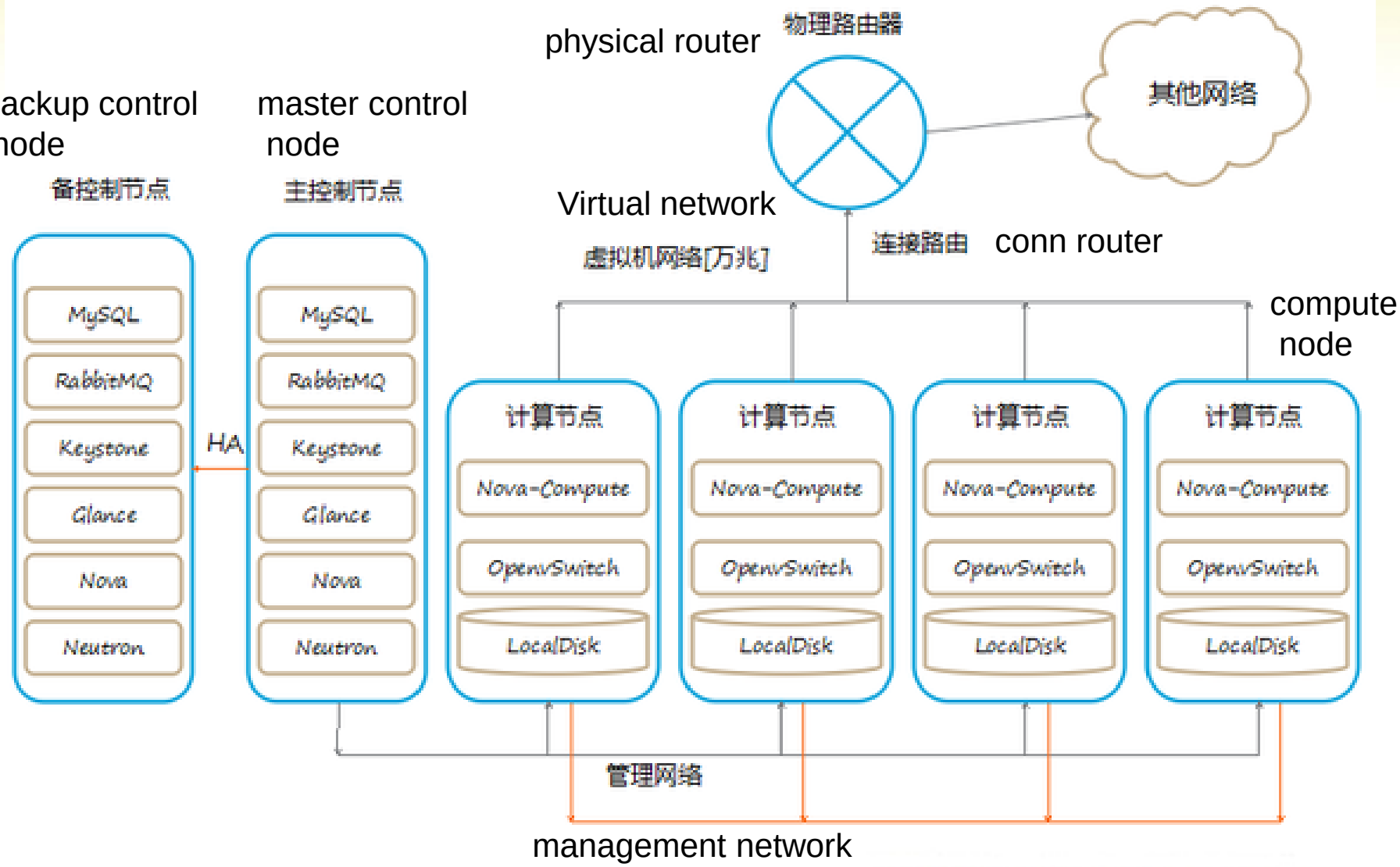
- IaaS compute platform
- Processing big data with tools like Hadoop
- Scaling compute up and down to meet demand for web resources and applications
- High-performance computing (HPC) environments processing diverse and intensive workloads
- Different users of OpenStack are AT&T, HP cloud services, PayPal, Rackspace, Sony online gaming systems, Yahoo, Wikimedia labs, Intel, NASA etc...

Private Cloud of 

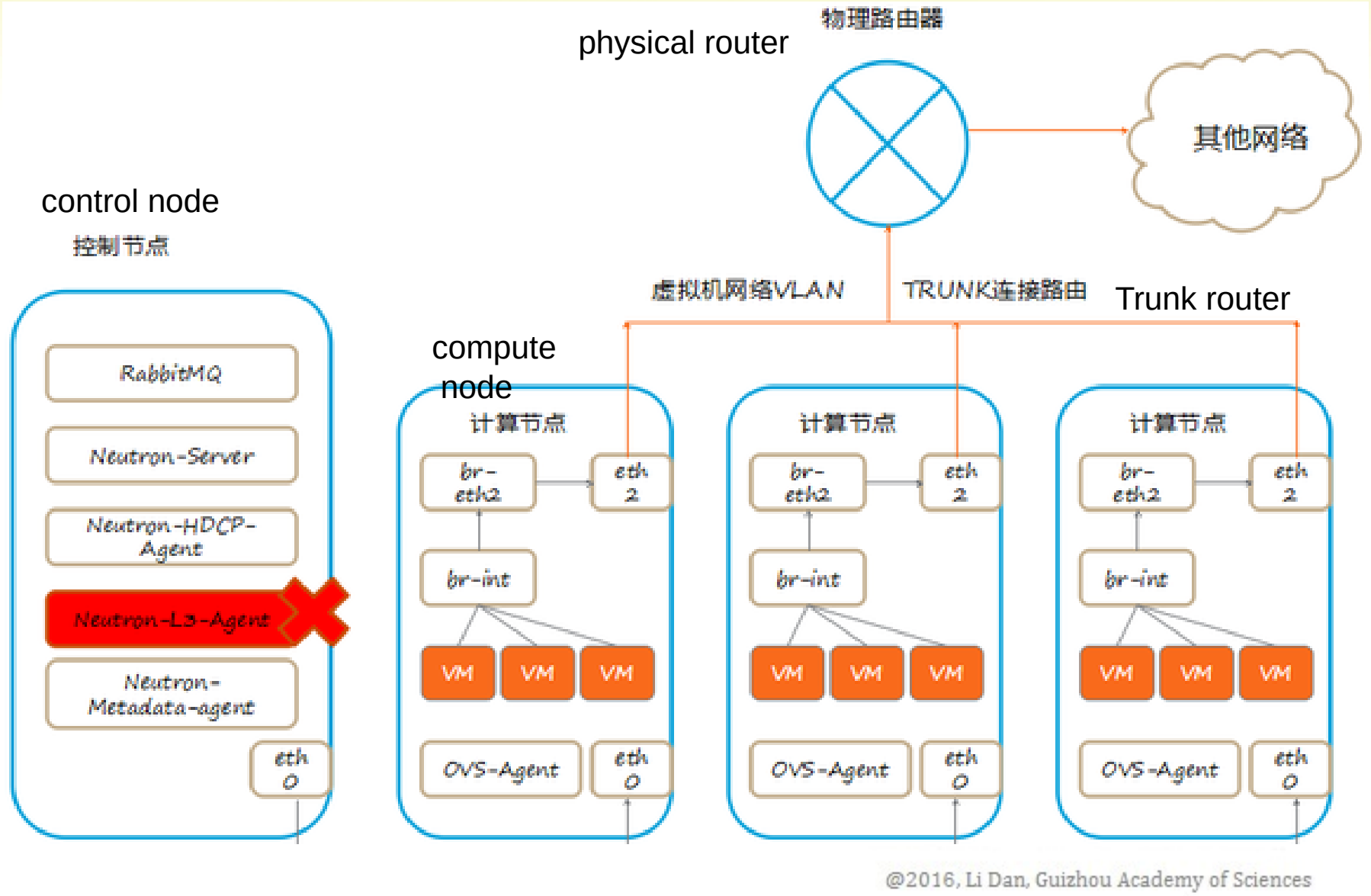
- Availability: 99.99%, 2 breakdowns in 16 months
- Utilization rate: 40%(physical). 1 PM : 12 VM
- The cloud spreads across 4 machine rooms. It has more 2000 VMs, 4500 physical cores (E5-2640), 50 T memory, 1200T virtual disk, 480 T block storage, 120 T object storage.

http://www.csdn.net/article/2015-06-29/2825070

Architecture of MI cloud



Details of the Architecture



More Facts about MI Cloud

- Compute node: DELL _R720
  - CPU： E5-2640v2*2(32 cores)
  - MEM：16G*24
  - Disk：2*600G SAS（Raid1) + 6*4T（Raid5） SATA
  - Net Card: 1G * 2 + 10G*2 (Intel 82599EB 10-Gigabit SFI/SFP+)
- Control node: DELL_R620
  - CPU： E5-2630v2*2 (24cores)
  - MEM：16G*4
  - Disk：2*600G SAS（Raid1) + 2*240G SSD(Raid1)
  - Net Card: 1G * 2 + 10G*2 (Intel 82599EB 10-Gigabit SFI/SFP+)
- OS : CentOS + RDO

Walmart is powered by 140,000+ OpenStack Cores

"Last Thanksgiving, we did 1.5 billion page views…November and December are the two months where we have to satisfy all the needs of all the customers, and that's what we did – we built about 140,000 cores"

-Amandeep Singh Juneja, WalmartLabs

PayPal’ s transactions powered by OpenStack!

Havana
> 12k hypervisors
> 300k cores
10+ availability zones
15+ virtual private clouds
> 1.6 pb block storage
100% KVM
100% OVS

Spring 2015

During the holidays for 2014, we were running 66% of PayPal front and mid-tier on OpenStack. That number is now 100%. **If you have doubts about OpenStack running in production, this is the proof point for you to say that OpenStack is doing financial transactions.**” -Subbu Allamaraju, eBay Inc

DOCKER

Docker

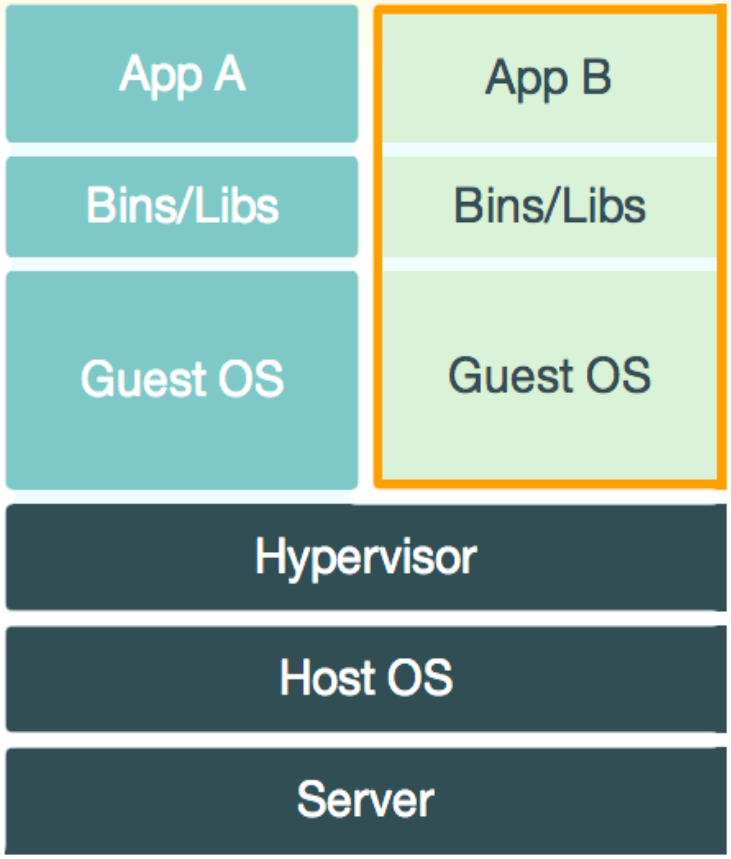
Docker is an open-source project that automates the deployment of applications inside software containers, by providing an additional layer of abstraction and automation of operating system–level virtualization on Linux.

[Source: en.wikipedia.org]

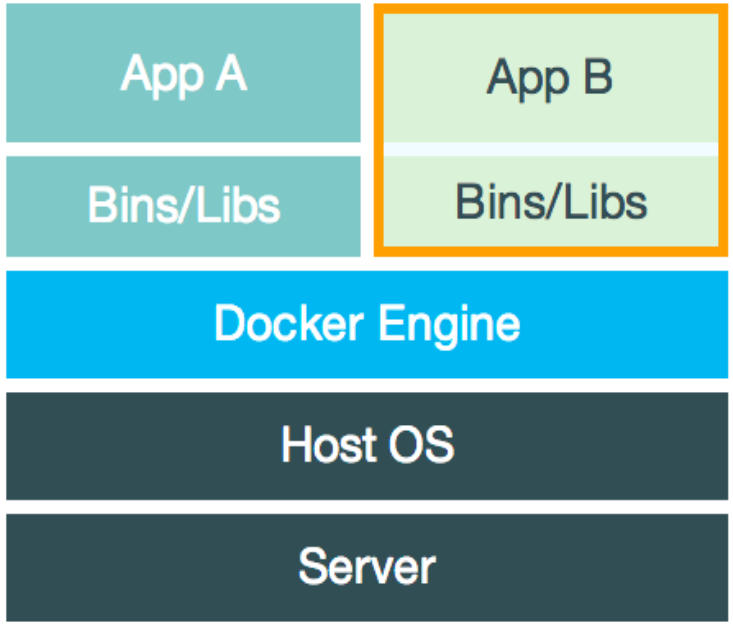
What is Docker?

- Open source platform of light weight container technology for virtualization
- Docker allows you to package an application with all of its dependencies into a standardized unit for software development
- Applications:
  - Are dockerized and
  - Run on Docker Containers
    - from laptops to production servers on cloud
  - Their images are shared on Docker Hub
  - Apps can be linked (node -> mongo)

Docker vs. Virtual Machine



Virtual Machine



Container

Docker vs. Virtual Machine

- VMs require apps to be contained within the guest OS
  - Potentially 1:1 on VM:application
- Docker sits between the app and the host OS
  - Takes place of the hypervisor
  - Runs images as an isolated process
  - Only the app and its libraries are compartmentalized instead of requiring the guest OS share that space as well

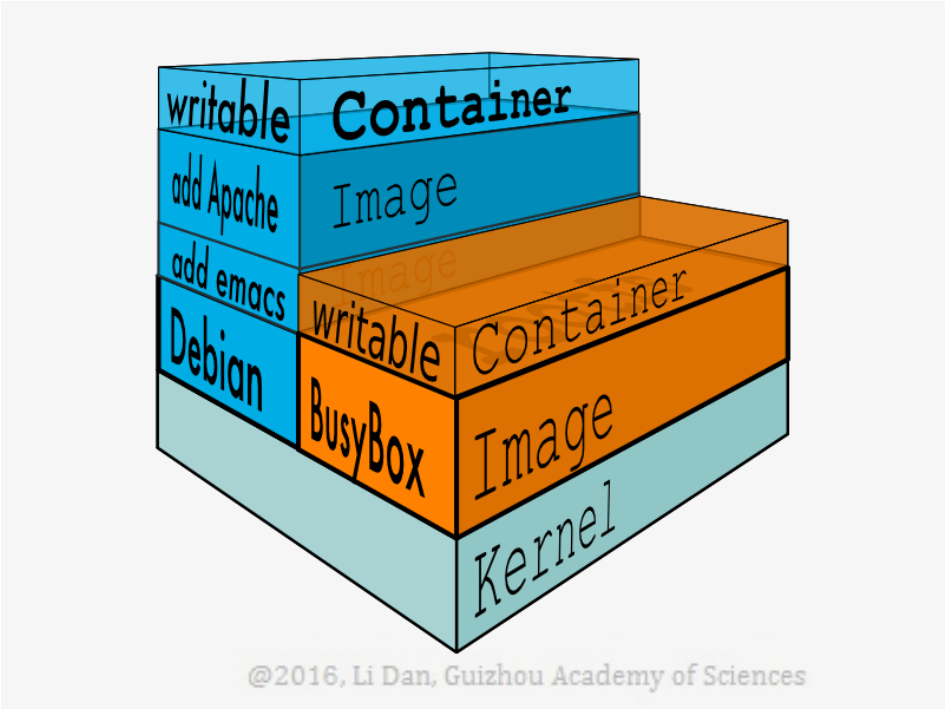
Why Linux Containers (LXC)

- Fast
  - Runtime performance near bare metal speeds
  - Management operations (run, stop , start, etc.) in seconds / milliseconds
- Flexible
  - Containerize a “system”
  - Containerize “application(s)”
- Lightweight
  - Just enough Operating System (JeOS)
  - Minimal per container penalty
- Inexpensive
  - Open source – free – lower TCO
  - Supported with out-of-the-box modern Linux kernel
- Ecosystem
  - Growing in popularity
  - Vibrant community & numerous 3rd party apps

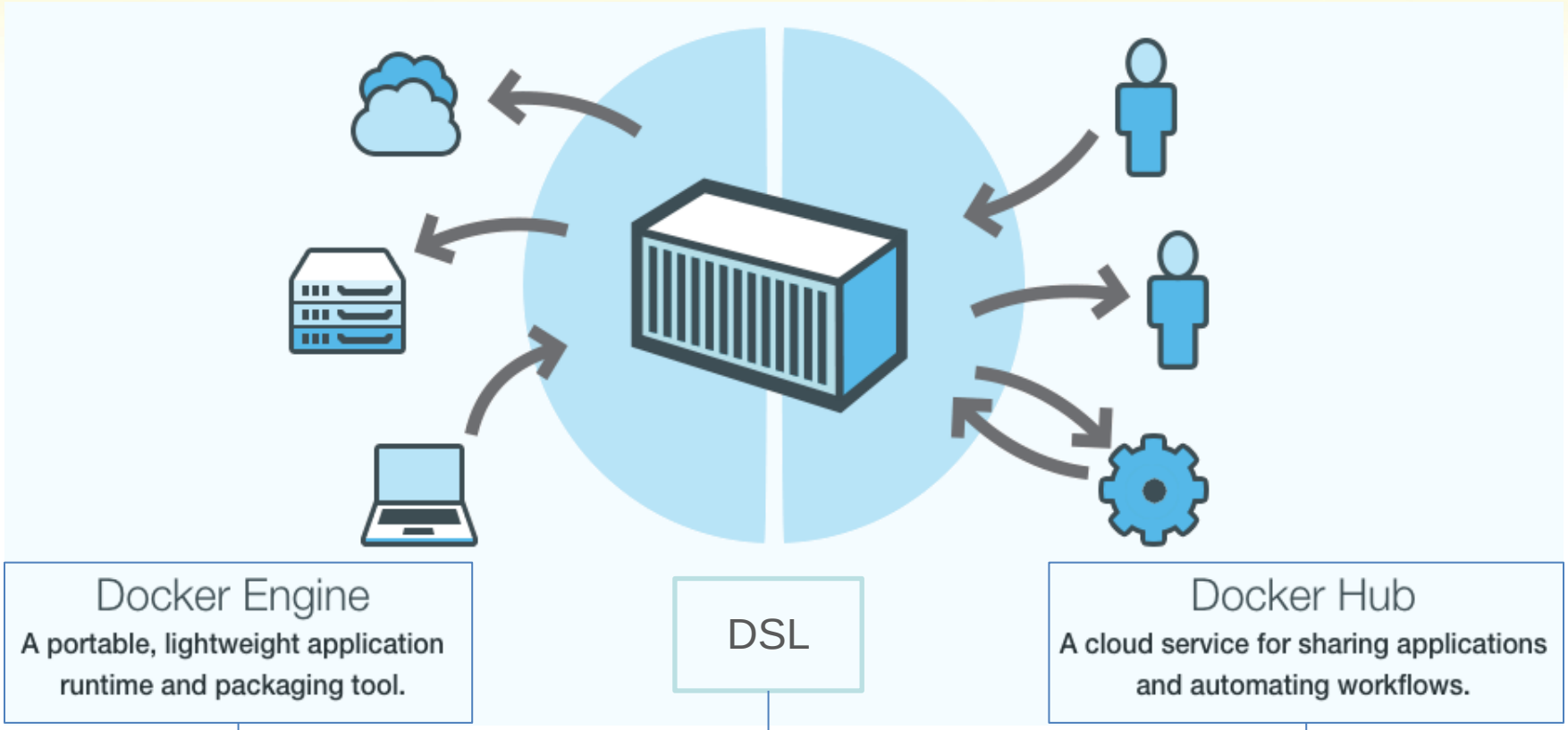
Containers Change the Game

Docker Technology

- libvirt: Platform Virtualization
- LXC (LinuX Containers): Multiple isolated Linux systems (containers) on a single host
- Layered File System



Docker Components



Docker Parts

- Docker Engine
  - Client-server application
  - Handles running images, registries and containers
- Docker Hub
  - Public Docker registry
  - Holds a collection of existing images to be used publicly
  - Image examples: Ubuntu, Node.js, MongoDB, WordPress

Docker Hub

- Public repository of Docker images
  - https://hub.docker.com/
- Integrates with Github and Bitbucket
  - Operates similar to them as well
- Hosts images that can be used and contributed to by the Docker community
  - Location that Docker pulls and pushes to

Dockerfile

- Docker can build images automatically by reading the instructions from a Dockerfile.
- A Dockerfile is a text document that contains all the commands you would normally execute manually in order to build a Docker image.
- By calling docker build from your terminal, you can have Docker build your image step by step, executing the instructions successively

Example Dockerfile

```
# Dockerfile to create a node image
FROM google/nodejs
MAINTAINER Carrotzpc
# Add files to the image
RUN mkdir -p /opt/nodejs
ADD . /opt/nodejs
WORKDIR /opt/nodejs
# Install the dependencies modules
RUN npm install
# Run make build
RUN make build
# Expose environment variables
ENV MONGO_CARROT_ADDR **LinkMe**
ENV MONGO_CARROT_PORT **LinkMe**
ENV MONGO_CARROT_DATABASE admin
ENV MONGO_CARROT_USER **ChangeMe**
ENV MONGO_CARROT_PASS **ChangeMe**
# Expose the container port
EXPOSE 5000
ENTRYPOINT ["node", "app.js"]
```

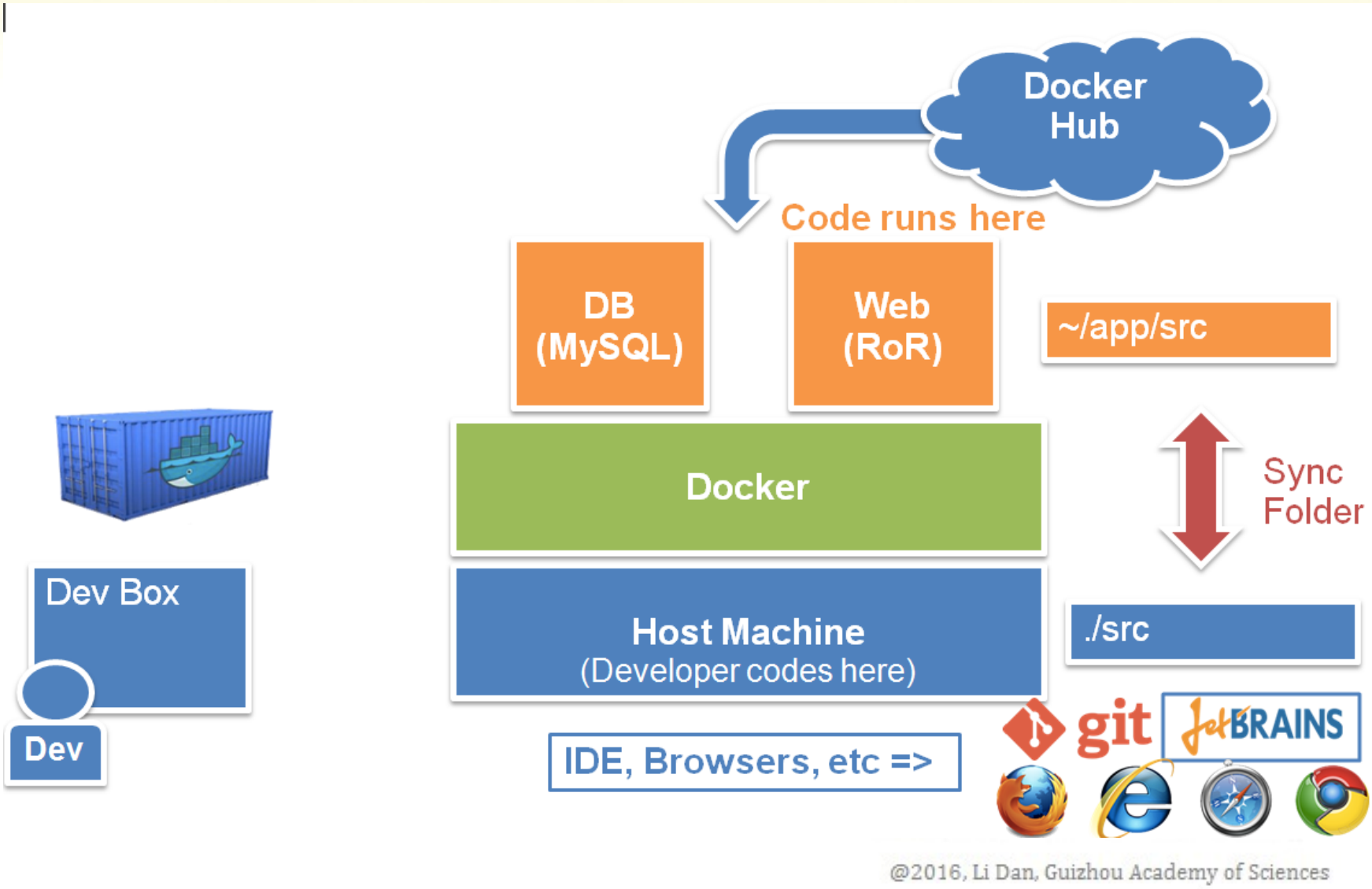
Runing Platforms

- Various Linux distributions (Ubuntu, Fedora, RHEL, Centos, openSUSE, ...)
- Cloud (Amazon EC2, Google Compute Engine, Rackspace)
- Microsoft plans to integrate Docker with next release of Windows Server

Running a Web App in Docker

- Similar to any other *nix command line application
- Docker's example runs a Python Flask app:
  - `$ sudo docker run -d -P training/webapp python app.py`
    - -P: map required network ports inside the container to the host
  - Check results with `sudo docker ps -l`:
    - `CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES bc533791f3f5 training/webapp:latest python app.py 5 seconds ago Up 2 seconds 0.0.0.0:49155->5000/tcp nostalgic_morse`
    - -l: tells `docker ps` to return the last container started

Development Scenario



MeanStack Use case

Open-Source Full-Stack Solution for MEAN Applications

MeanStack

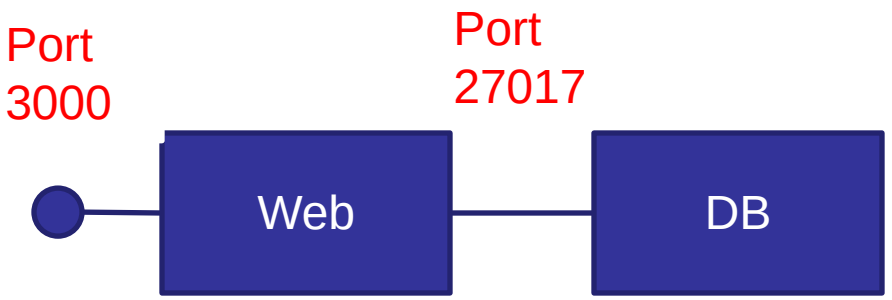
- Mean.js provides
  - Code generator to generate Mean App
  - Mean.js apps typically have
    - Node Js Server
    - Mongo DB database
  - Provides Dockerfile and fig.yml to run the app in Docker Containers
    - One Docker container for Node Js Server
    - One Docker container for Mongo DB Database

MeanStack Configuration

fig.yml

```
web:
  build: .
  links:
   - db
  ports:
   - "3000:3000"
  environment:
   NODE_ENV: development
db:
  image: mongo
  ports:
   - "27017:27017"
```

fig.yml



Docker Containers

Who uses Docker?

Docker PAAS Providers

And many more…

Who uses Docker?

As an Infrastructure Tool along side

Docker Integration

- OpenStack + Docker
- Hadoop + Docker
- Spark + Docker
- Mesos + Docker
- ……

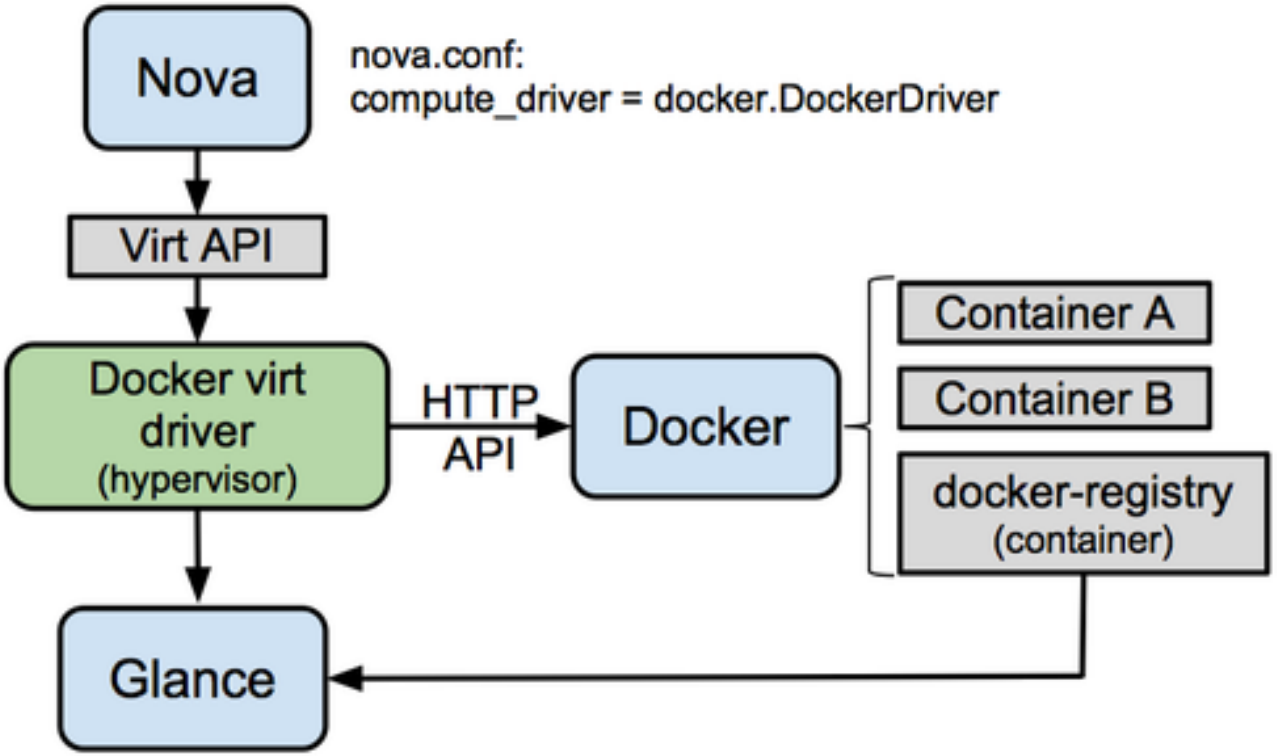
OpenStack Deployment With Docker

- **Nova:** add a new nova docker driver. https://github.com/stackforge/nova-docker
- **HEAT:**
  - Add a new HEAT Resource：
  - HEAT docker driver interacts with docker server
- **Container as a Service**
  - A new serice for OpenStack for manage docker container

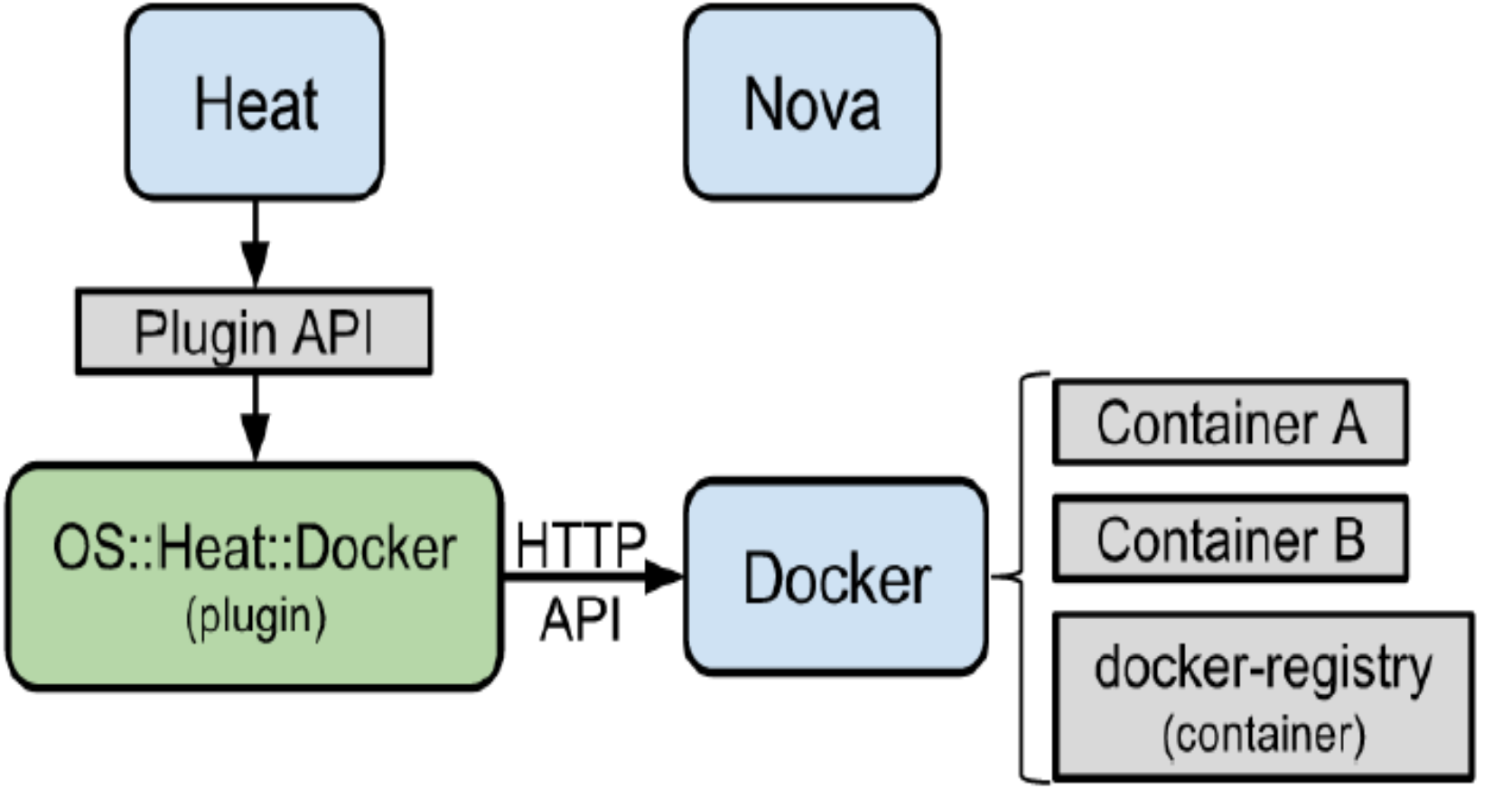
Solutions are still under development

Havana: **nova-docker driver**

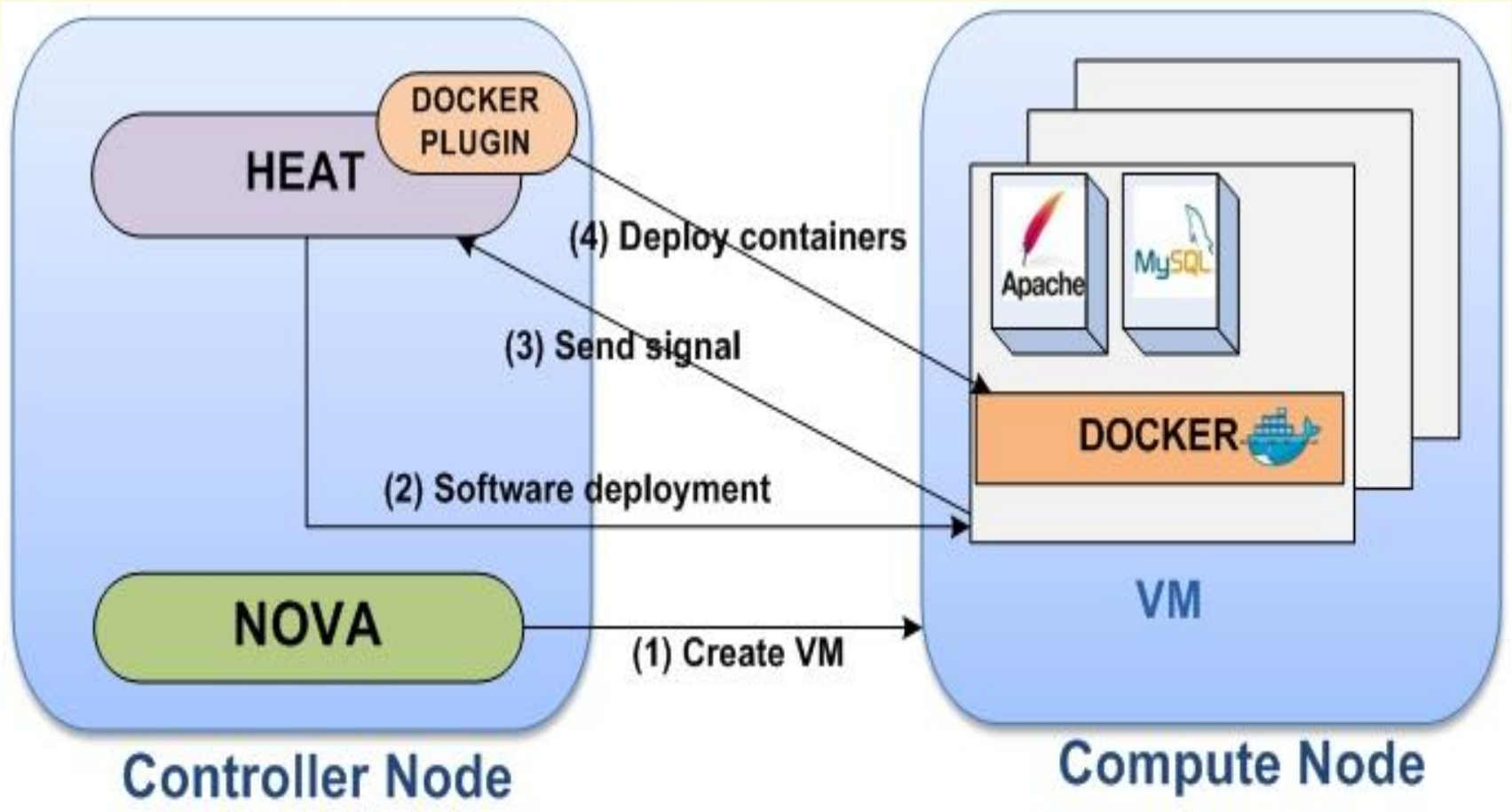
- Nova driver which integrates with docker REST API on backend
- Glance translator to integrate docker images with Glance



Icehouse: **heat plugin for docker**



HEAT Docker Driver



- https://github.com/MarouenMechtri/Docker-containers-deployment-with-OpenStack-Heat
- http://techs.enovance.com/7104/multi-tenant-docker-with-openstack-heat