Thomas Spatzier (thomas.spatzier@de.ibm.com)
Cloud & Smarter Infrastructure CTO Office

Lakshminarayanan Renganarayana (lrengan@us.ibm.com)
IBM Research Staff Member

Heat and Enterprise Applications

OpenStack Summit Hong Kong, November 2013

About this session

In this session we want to talk about:

Application orchestration: the deployment of application components, including their underlying infrastructure, as well as management of applications and their infrastructure throughout their lifetime

… for **enterprise applications**: typically larger scale deployments with higher requirements on scalability, reliability and performance

We want to share experiences from two solutions:

- Application orchestration in IBM SmartCloud® Orchestrator, based on the OASIS TOSCA standard
- Weaver, a higher level DSL aimed at DevOps scenarios and continuous delivery

We want to explore relationships to OpenStack Heat:

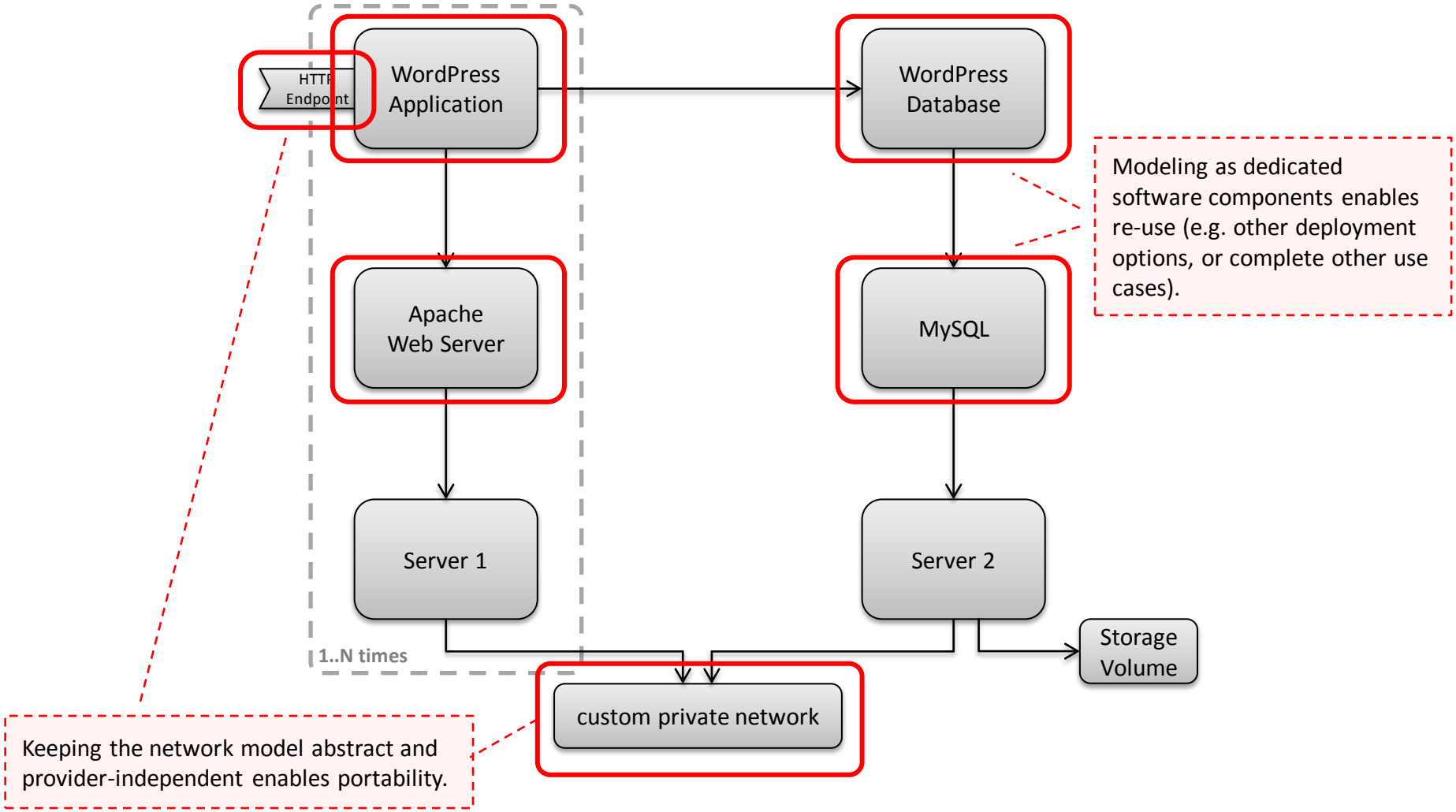
Current functionality, ongoing discussions, potential future directions

Agenda

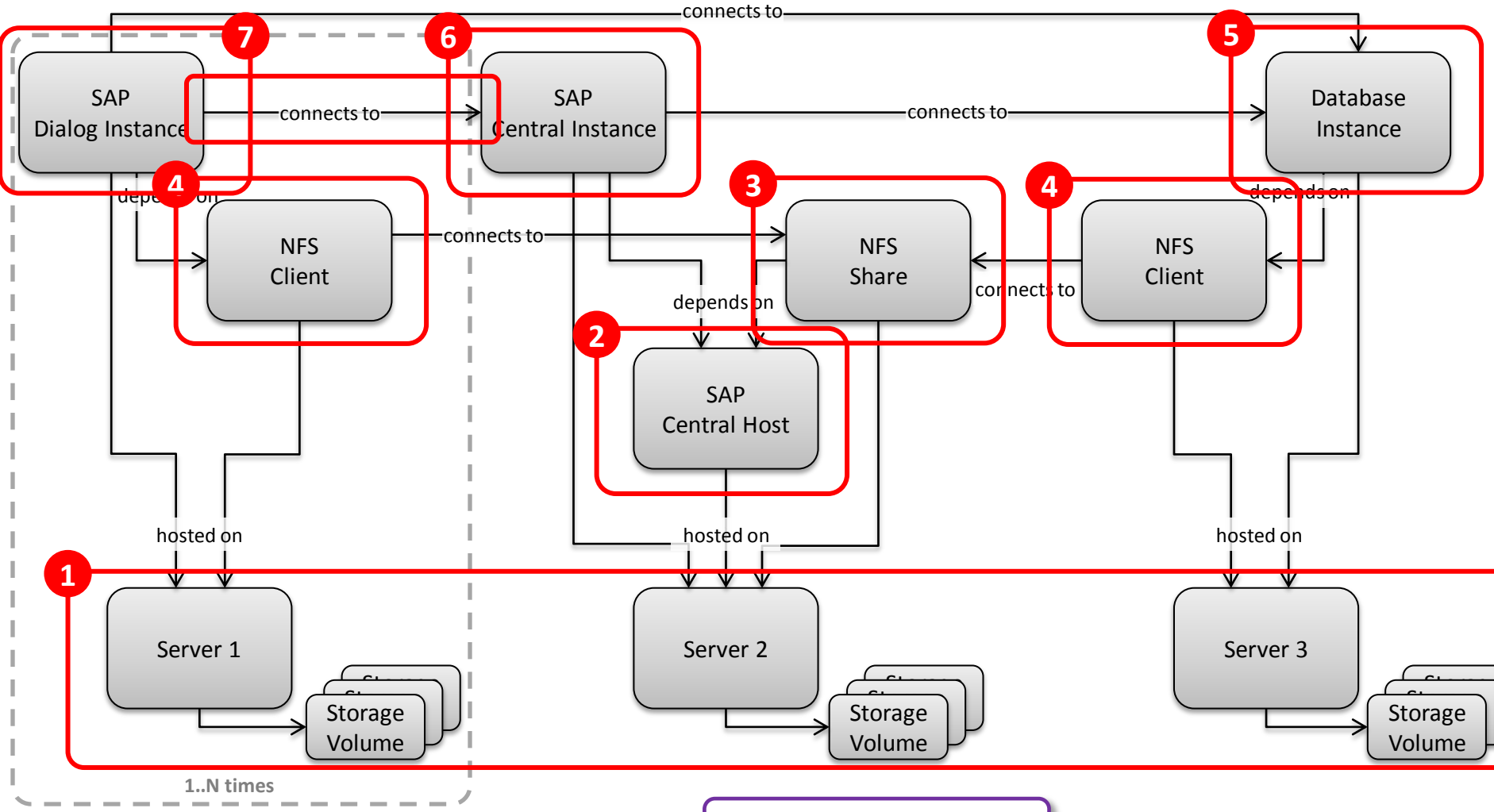
- Some examples
- Common requirements on software orchestration
- Current solutions … and their use of Heat
- Ongoing activities in the Heat community

Some examples

***Getting started – WordPress**: two-tier WordPress deployment with some infrastructure requirement. Dedicated software component modeling enables **re-use**. Abstract network model ensures **portability**.*

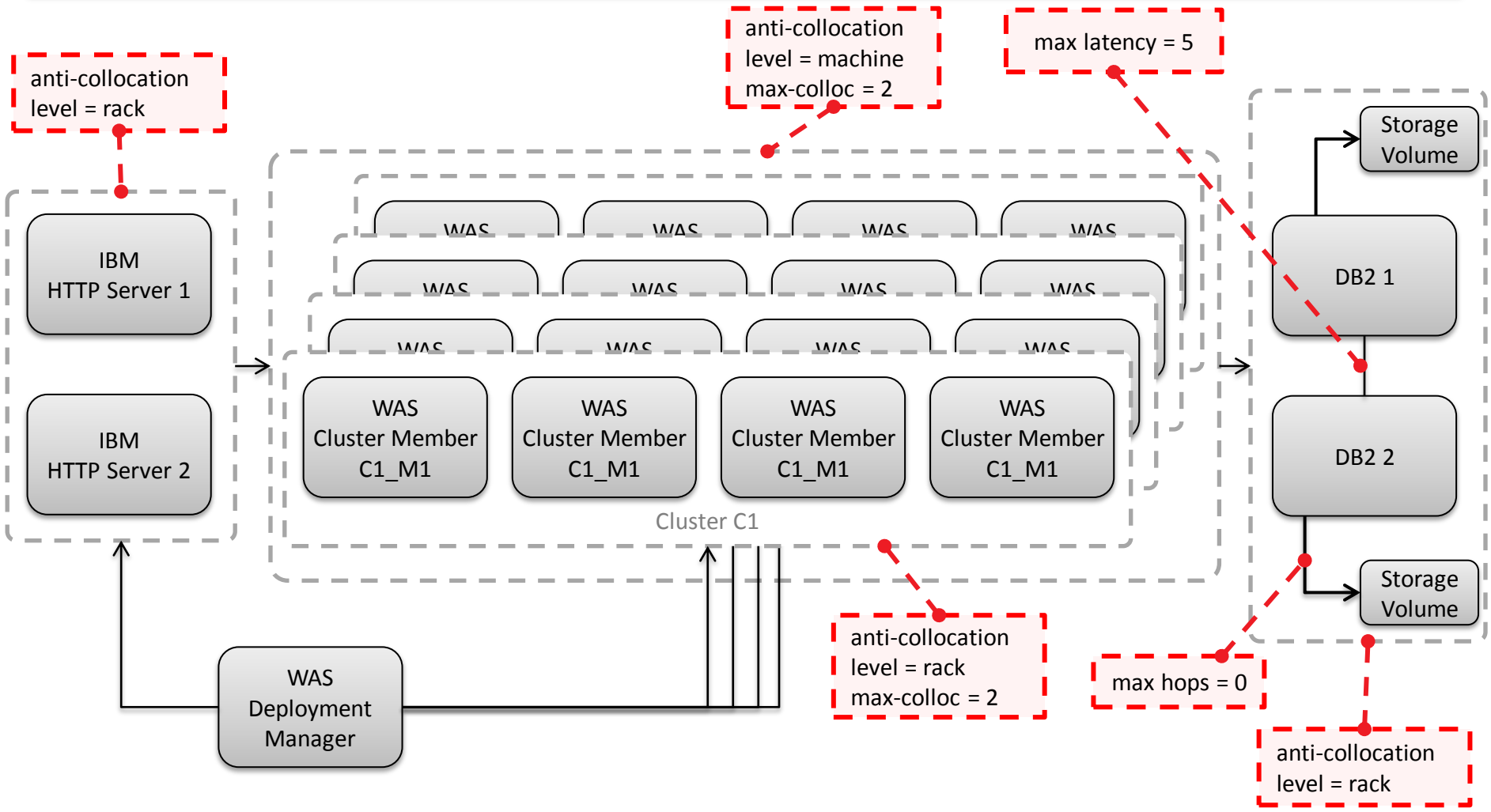


***Multi-tier SAP application**: many software components spread across multiple VMs, with many dependencies between the components. **Processing flow** can be **derived based on relationship graph**.*



powered by OASIS TOSCA

***IBM collaboration platform**: highly scalable application with high requirements on availability. Special **placement policies** to support availability and **connectivity policies** to support performance goals.*



Common requirements on software orchestration

Common requirements on software orchestration

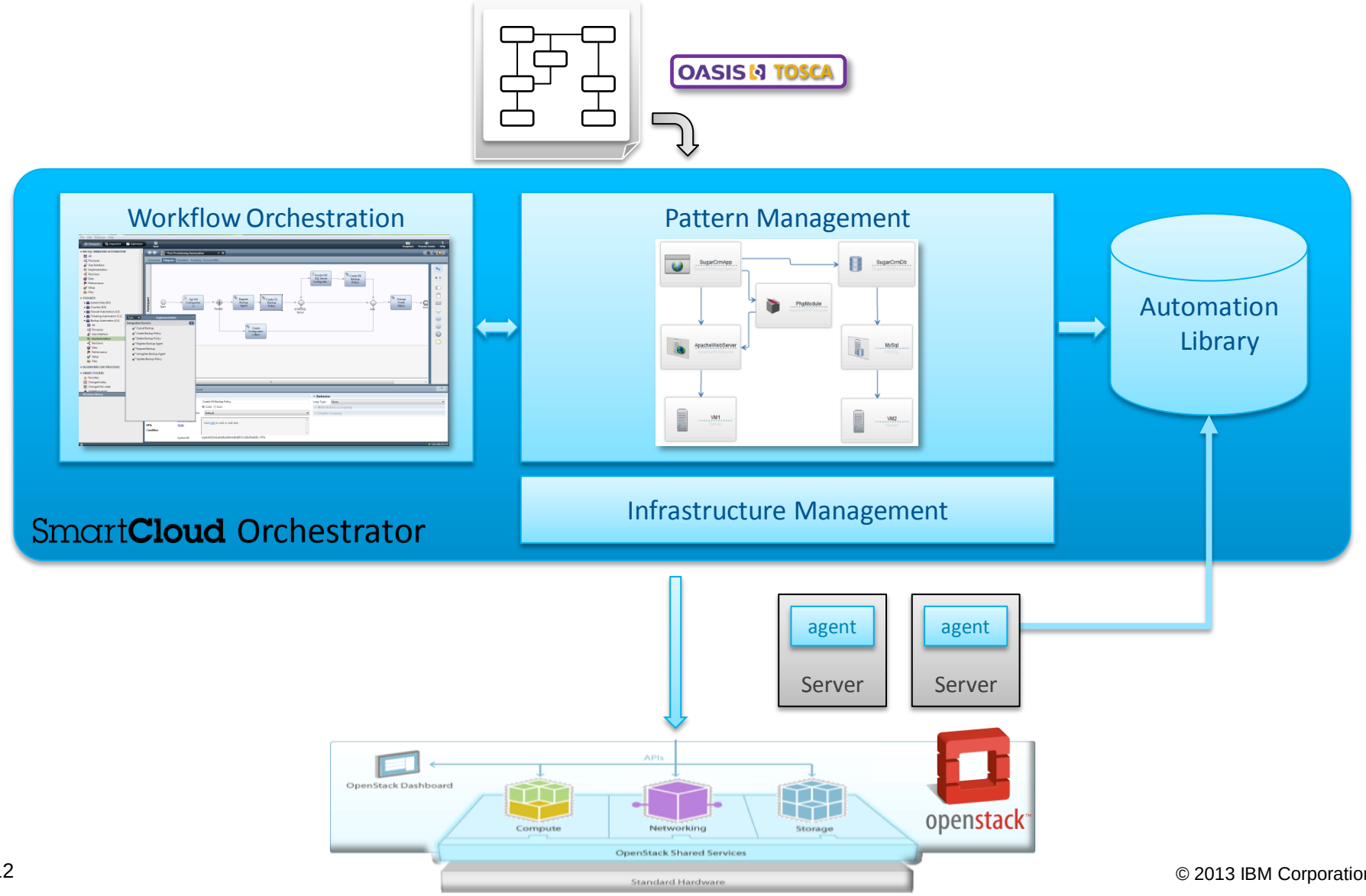
- Proper representation of **software components in models**
  - Ability to re-use components in other contexts
  - Representation of stateful entities with properties that can be set and observed, and with a runtime state
- Expression of **dependencies** between components with well-defined semantics to derive the proper **processing flow**
  - Handling of multi-instance components: process all in parallel or one by one, apply special naming
- Ensure **portability** by defining as little as possible but as much as necessary about an application's infrastructure
  - The concrete infrastructure can look different in each environment, but an application pattern should be re-usable across environments
- Ability to express special requirements on placement (**"policies"**) to meet non-functional requirements

Beyond initial deployment

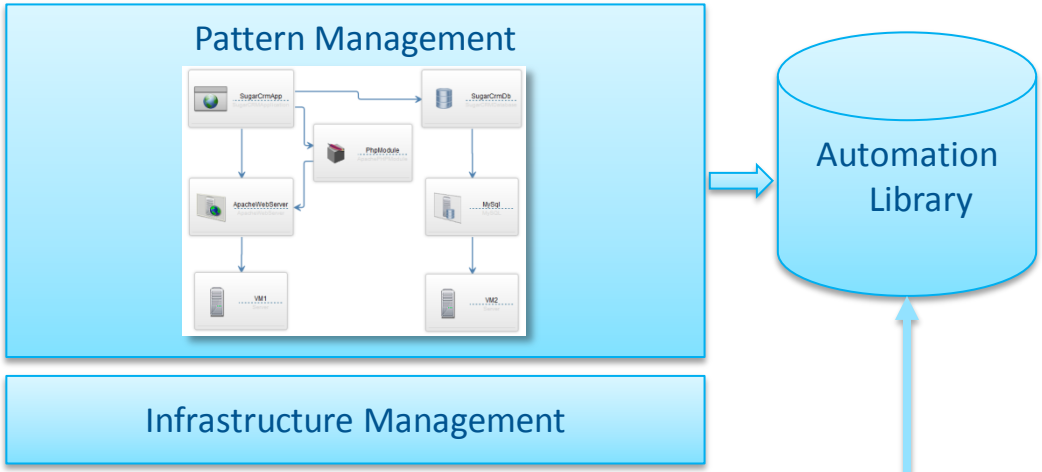
- Deployment is only a fraction of enterprise applications' **life time**
- **Scaling** based on infrastructure or **application metrics**
  - Infrastructure metrics: infrastructure drives the application
  - Application metrics: application drives the infrastructure
- Scaling, failover and other changes to a deployed topology require **proper handling**
  - Need ability to hook in automation to update the application layer
- **Updates** to long running applications must be possible, ideally online
- Complete **custom flows** based on operations provided by application components of a deployed application (DB backups, patching, maintenance, …)
  - Workflows can run on top of a pattern engine based on a proper interface

Current solutions
… and their use of Heat

Software orchestration in IBM SmartCloud Orchestrator – Overview

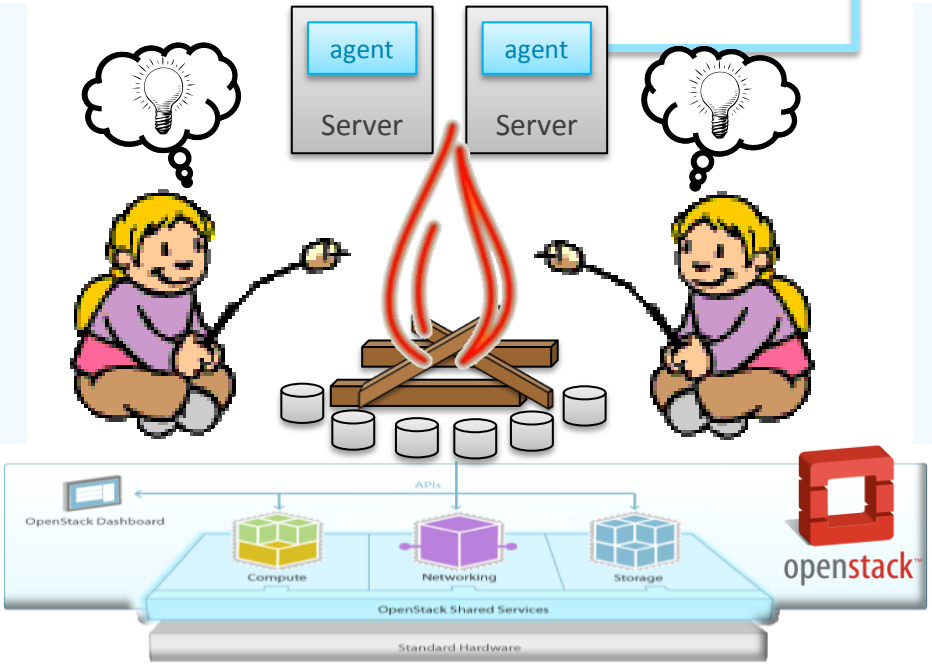


Handling of software defined infrastructure



Infrastructure Management layer:

- Requires previous setup of
  - Networks and IP groups
  - Storage pools
- Allocates resources in scope of those pools per deployment
- Orchestrates allocation flow of different resources

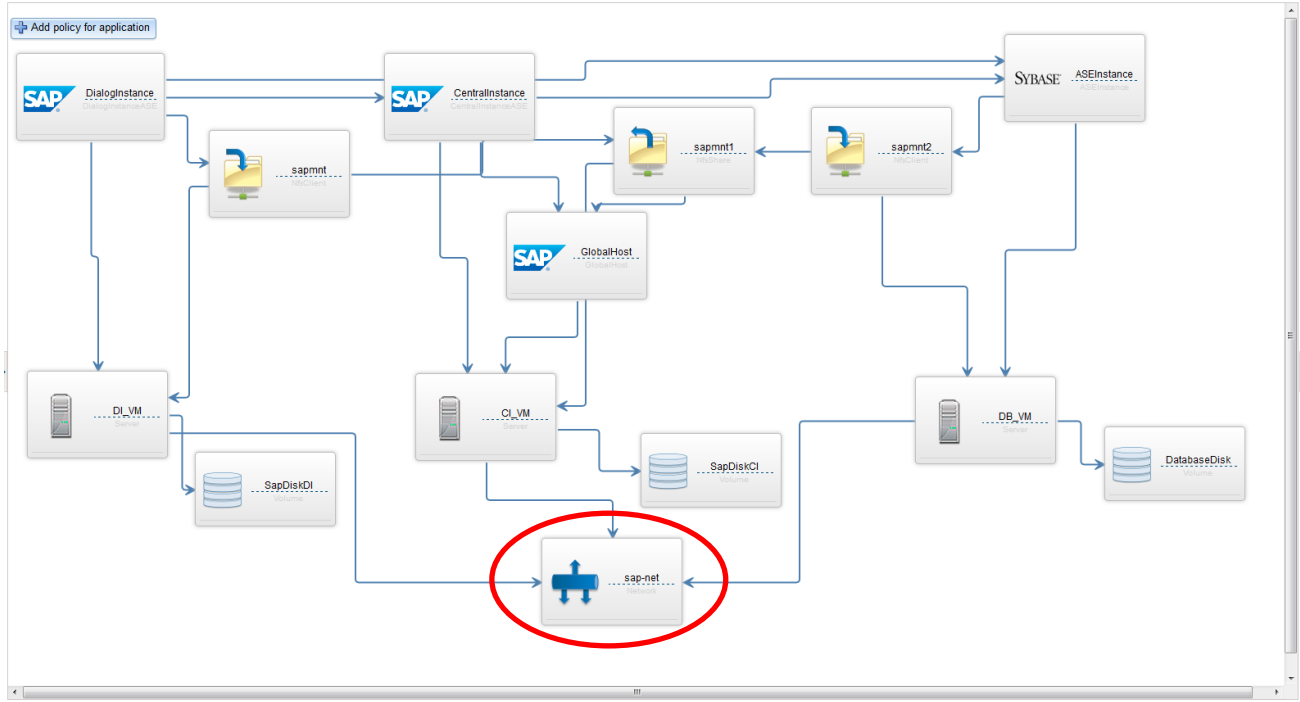


Adding flexibility to the infrastructure layer:

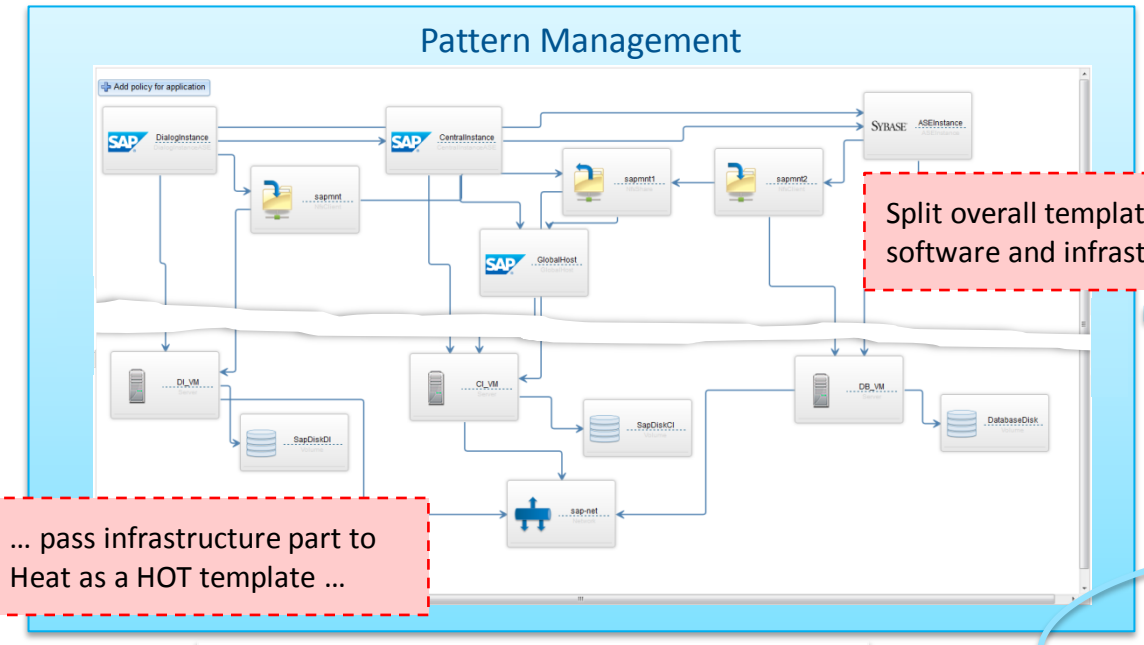
- More dynamic infrastructure configuration per pattern deployment
- Extended set of deployment topologies
- Extended set of resource types

Prototype use case: Multi-tier SAP CRM training system

- Deployment of multi-tier SAP CRM systems for training purposes
- Identical configuration (incl. SAP system ID) for each trainee to have consistent base setup
- SAP systems require network isolation to prevent system ID clashes
- Deploy each SAP system into its own new network



Prototype deployment flow

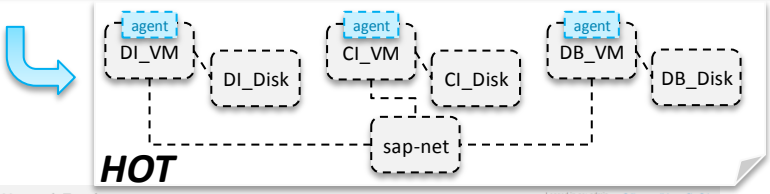


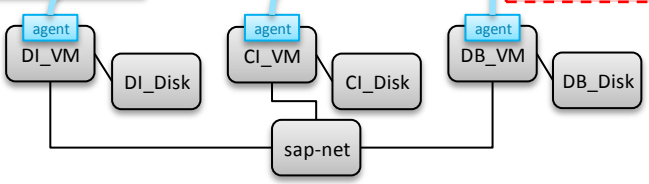
Split overall template into software and infrastructure …

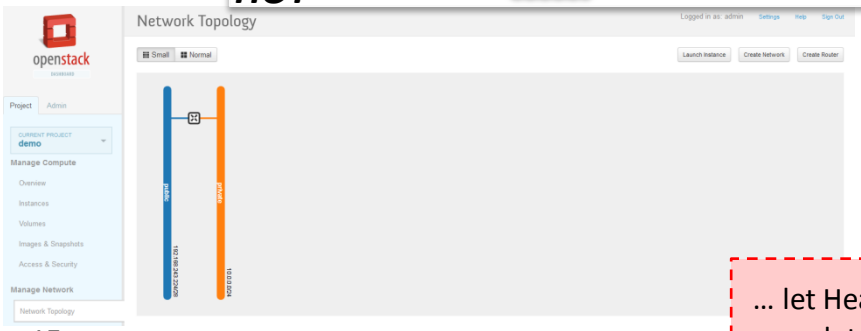
Automation Library

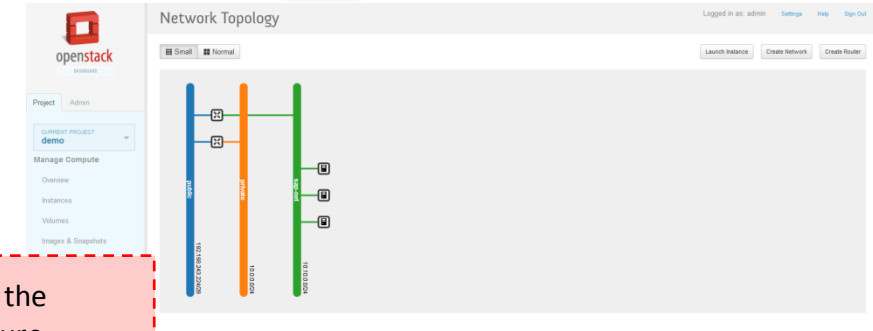
… pass infrastructure part to Heat as a HOT template …

… agents take over for software configuration



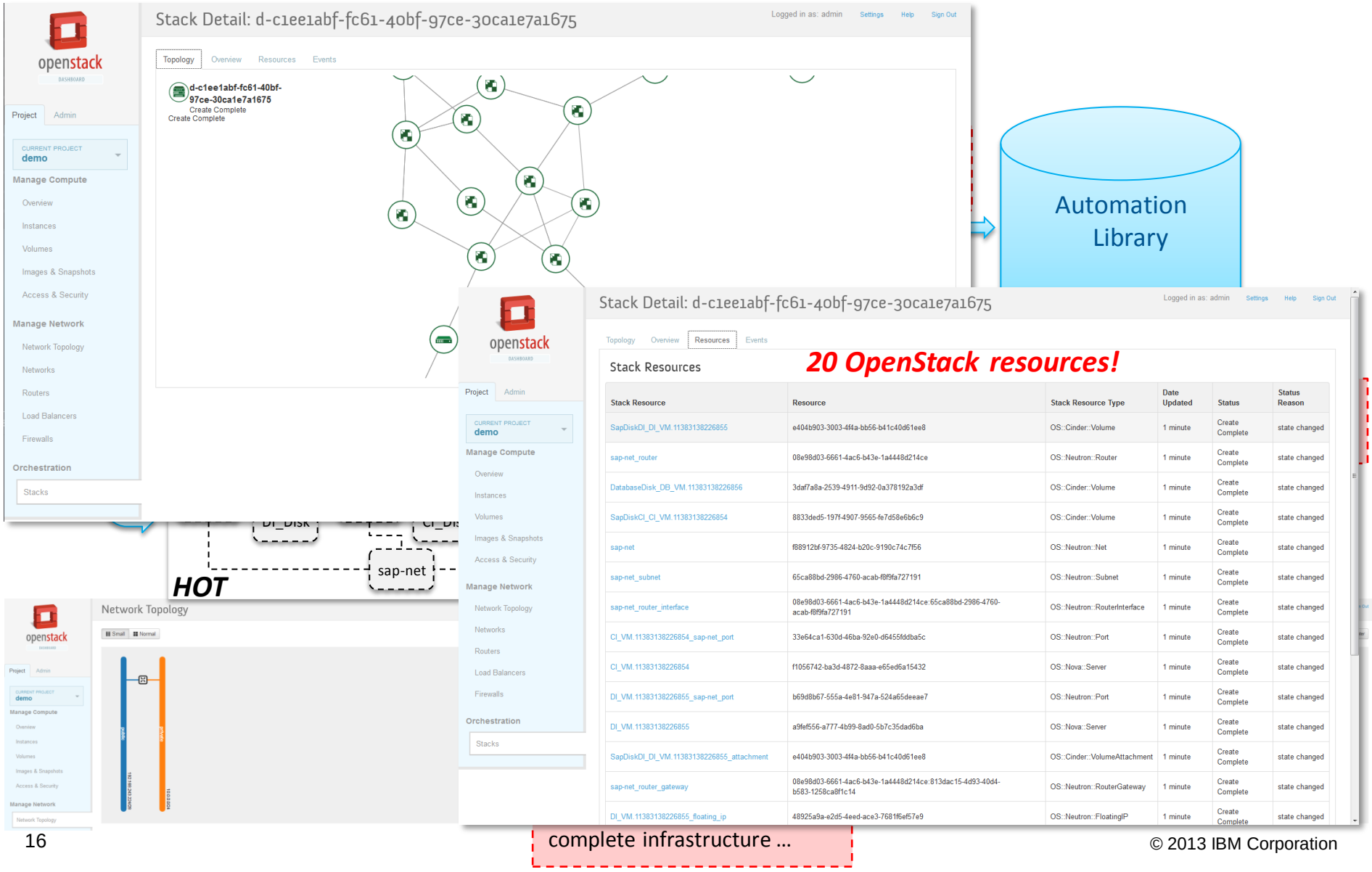






… let Heat configure the complete infrastructure …

Prototype deployment flow



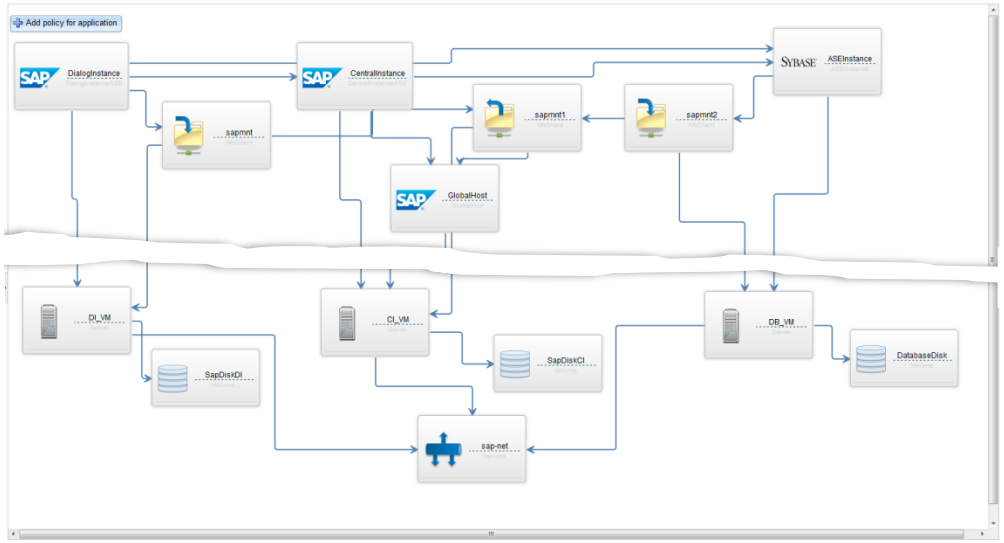
Observations from prototype

- Heat brings tremendous value add for complex infrastructure setup
  - Prototype use case required 10 Heat resources types with various dependencies for which we would have needed to implement support in our solution
  - Quick implementation time through hidden complexity
  - Processing offload to Heat
- A relatively simple pattern from user's perspective results in many infrastructure resources
  - Pattern portability across environments requires some abstraction of infrastructure
  - Heat provider templates can be used to map abstracted parts of environment specifics
- Agent bootstrapped as "software configuration provider" to handle software components
  - Possible since we have complete orchestration (dependency handling etc.) in our agent framework
  - With some component orchestration in Heat, this would also be possible for other providers (Chef, Puppet, scripts …)

Autonomic behavior at different layers …

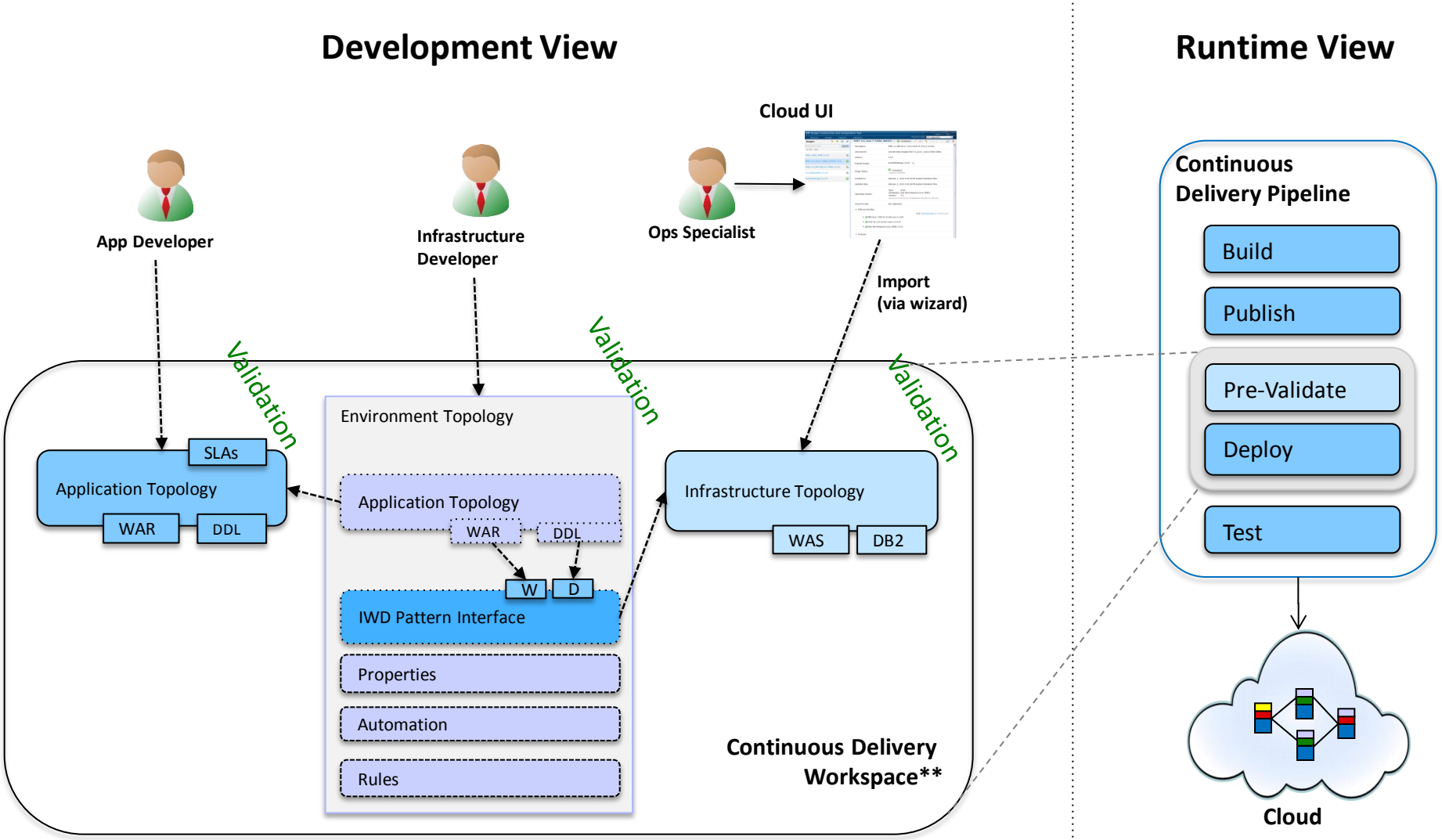
Agent framework
- Monitoring of application level metrics (transaction time, user sessions, etc.)
- Request infrastructure modification based on application level thresholds

Heat auto-scaling
- Monitoring of infrastructure metrics
- Infrastructure scaling based on monitored metrics



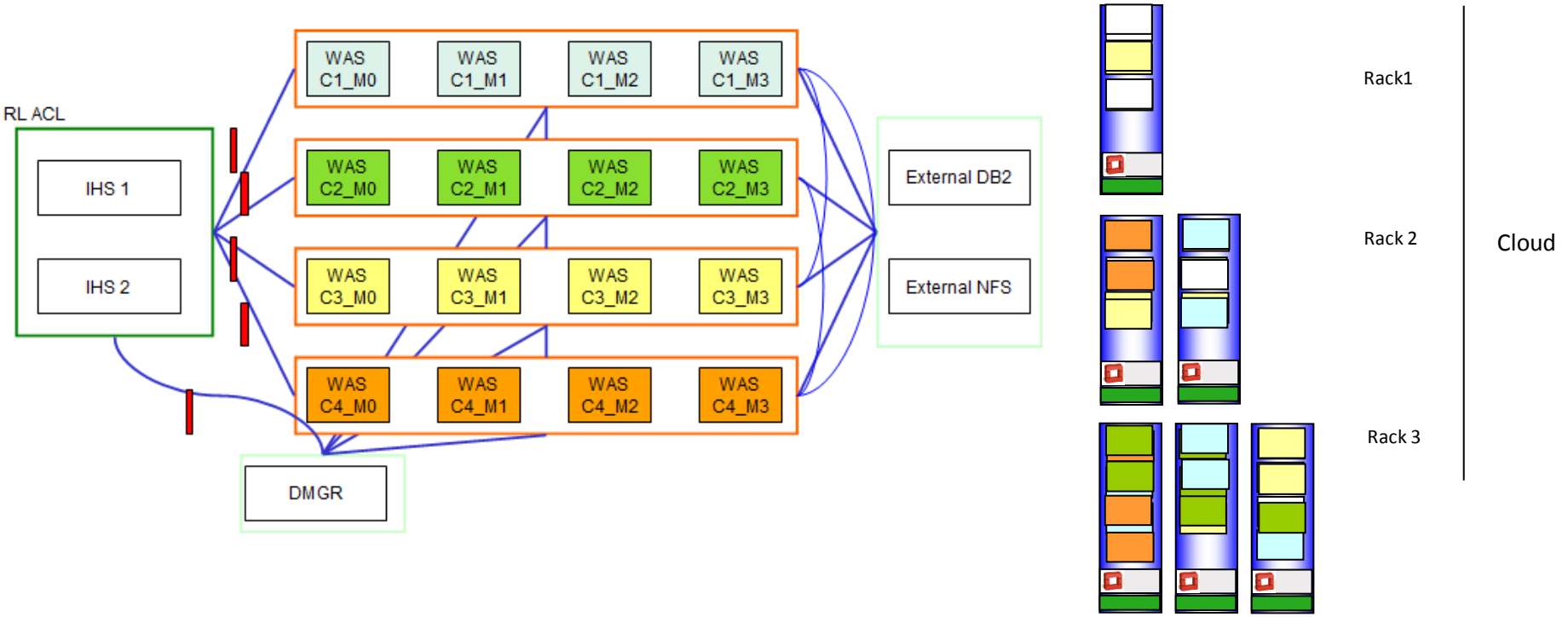
- Autonomic behavior in two layers causes trouble, or requires proper signaling (… which can become difficult)
- Scaling actions need proper handling in the application layer (e.g. this can mean more than just HTTP traffic load balancing)
- Possible solution: put software orchestration layer into driver's seat
  - No auto-scaling used in Heat
  - Heat stack update triggered by software orchestration layer

Weaver – a DSL for Continuous Deployment



**Versioned in SCM

IBM Connections – a social collaboration application used by 400 thousand IBMers everyday (profiles, blogs, wikis, activities)



VM group with anti-collocation policy: spread across at least 2 racks, no two VMs on the same physical machine

VM group with anti-collocation policy: spread across at least 2 racks

IBM Connections – a social collaboration application used by 400 thousand IBMers everyday (profiles, blogs, wikis, activities)

Provisioning environments is hard and error prone

- Complex multi tier application made up of Front end load balancers; IBM Websphere (active / passive) ; IBM DB2 ; Files storage ; email; Single Sign On
- Wiring of compute, network and storage resources, including software, virtual or physical platforms, network configuration and external services
- Availability, security, performance, and resource utilization are affected by layout on the infrastructure

Agility requirements

- Business criticality driving architectural considerations for HA are a top priority. Risk adverse while business needs rapidly changing.
- Complex application architecture coupled with deployment operations tribal knowledge slows new function roll out
- Release to release configurations are difficult to maintain and evolve
- Automation based on scripting techniques does not scale and too fragile, difficult to maintain and evolve due to implicit dependencies and environmental assumptions

Example specification using Weaver DSL

```
topology (:connections_pattern) {
  # create four clusters
  4.times do |i|
    # create a unique WAS cluster id symbol
    node ("was_cluster#{i+1}".to_sym) {
      # on each cluster create 4 nodes
      multiplicity 4

      include '../automation/connections_was_role.weaver'

      connections_was_role.wasadmin_password = config[:wasadmin_password]
      connections_was_role.dmgr_hostname = late_binding { was_dmgr.ip_address }

      redundancy_constraint (:rc) {
        spread_across_at_least({ :rack => 2 })
        all_different :compute_node
      }

      licensed_product(:p1){product_id '5724H88'}
    }
  end
}
```

Create four clusters using 4.times loop

Create four nodes in each cluster using multiplicity 4

directly leverage / reuse community recipes and Chef eco system

Evaluate this expression as late as possible → on the VM and propagate the value via an external coordinator

- Place this VM across at least 2 racks but no more than ceil(multiplicity/2) on same rack
- Each VM on a different compute node

Licensing constraint used by license optimization component

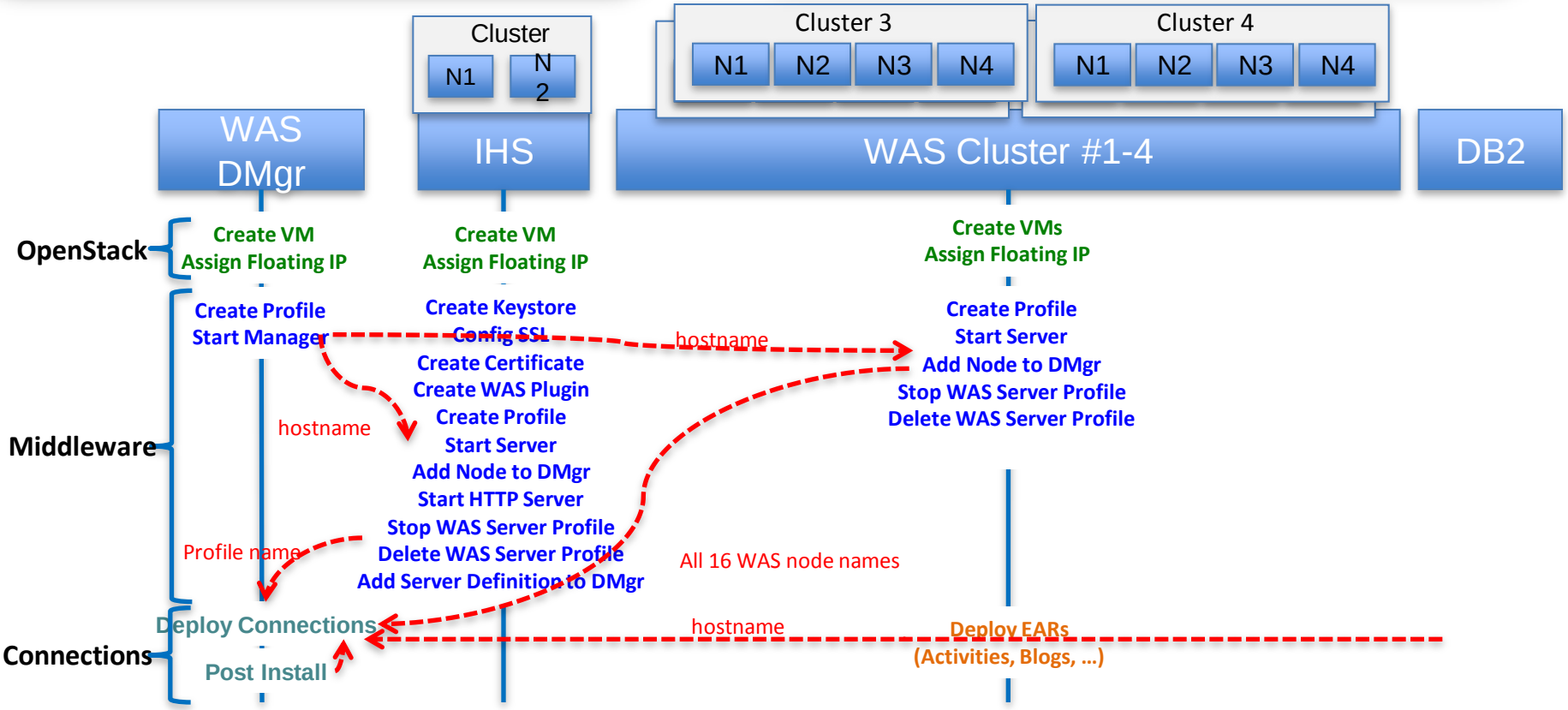
Cross component configuration of a distributed software system

Why it is complicated

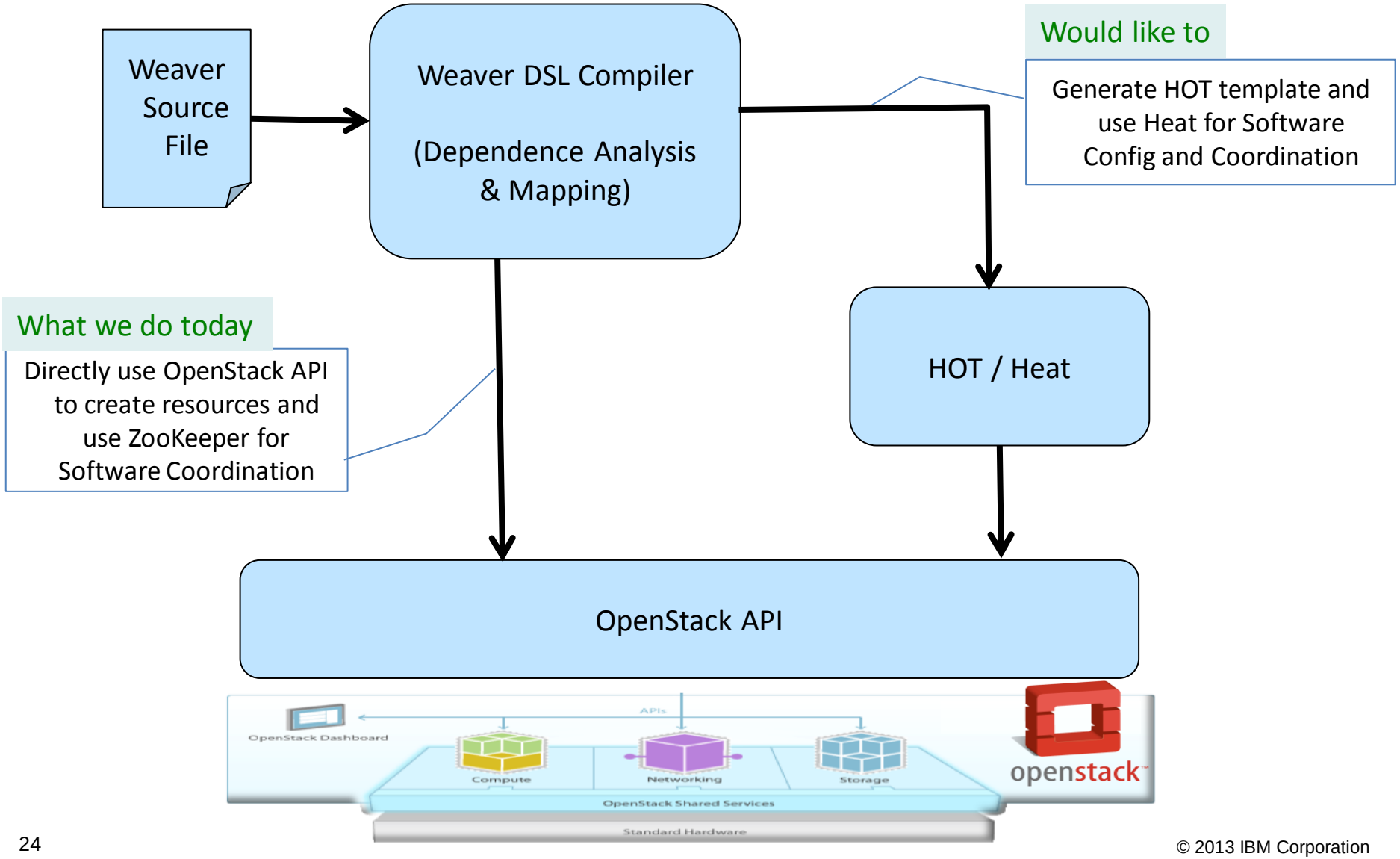
- Multiple fine grained configuration tasks
- Config tasks require certain order
- Data dependencies and data values available only during the deployment
- All buried as tribal knowledge

Weaver approach

- **Higher level language** – translate tribal knowledge to formal **repeatable** knowledge
- **Simplification**: focus on automating each granular step, runtime takes care of temporal dependencies and data passing



Deploying to OpenStack with Weaver



Ongoing activities in Heat community

HOT discussions around software orchestration

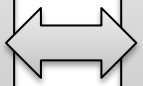
- Software orchestration has been one of the HOTtest topics recently in the Heat community and at the current design summit

- Key goals and design principles
  - From inlined **`user_data`** scripts to clearly defined software components
    → Clean separation of software from infrastructure
    → Better re-use of software component definitions
    → More flexibility in defining concrete deployment topologies
  - No duplication of software configuration technologies (Chef, Puppet, …)
  - User friendly template format for majority of use cases

- Two discussions

HOT constructs

- Definition of (software) components in HOT
- Declaration of data flow (component inputs and outputs)
- Declaration of dependencies between components (explicit and data flow based)
- …



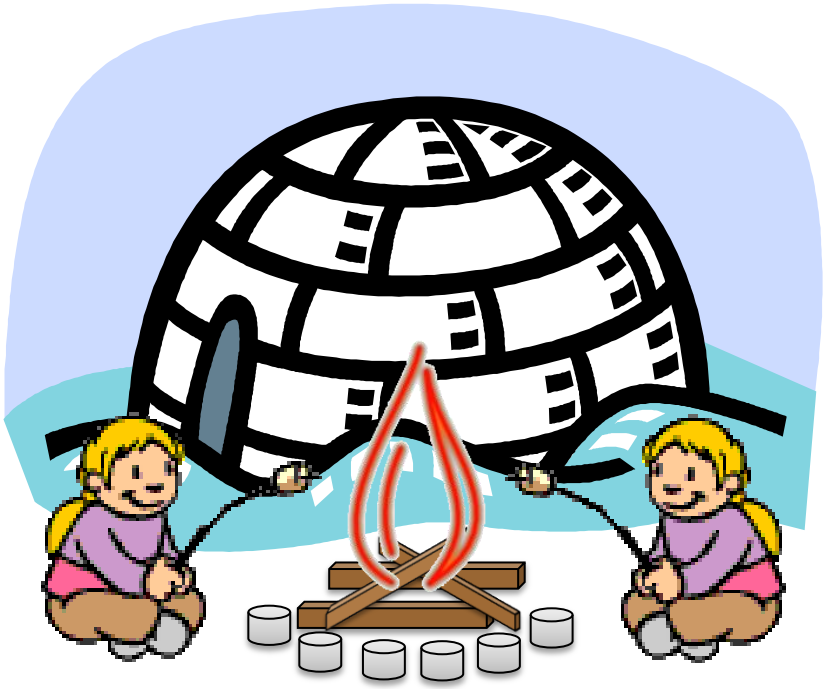
How to implement

- In-instance software configuration tool bootstrapping
- Metadata passing to software configuration tools
- Signaling (e.g. component completion) and data passing using existing mechanisms
- …

Discussions around policies, placement, …

- Enterprise applications for use in production environments bring in special non-functional requirements
  - Collocation / anti-collocation for failover and performance reasons
  - Placement for optimized communication paths
  - Placement optimized for license usage
  - …
- No Heat/HOT only discussion but requires close interlock with other projects
  - Intuitive definition of policies in HOT templates
  - Passing of policy metadata to underlying services (e.g. nova, cinder, neutron, …)
  - Enforcement of policies through underlying services
    (But where to handle cross-cutting aspects?)

Looking forward to an exciting Icehouse development cycle!



Legal Disclaimer

IBM