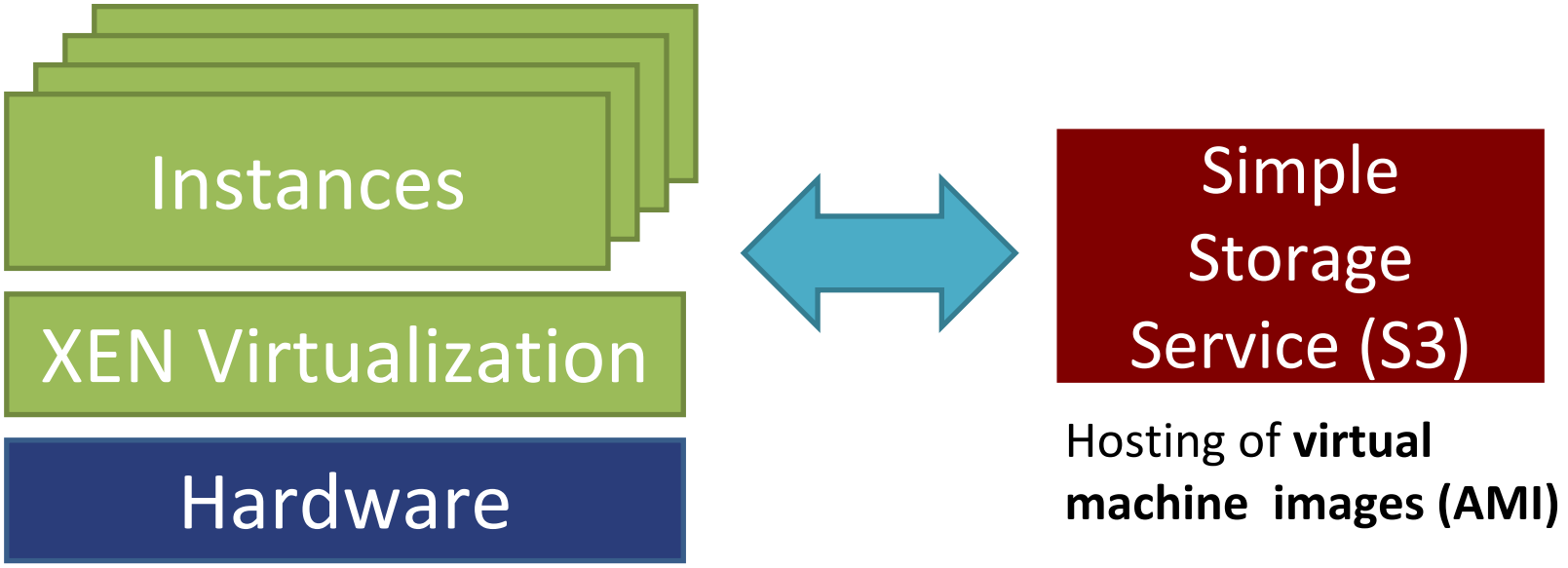
Implementation of TripCom Scalability Tests using Amazon EC2

Overview: What is EC2?

- Amazon EC2, **Elastic Compute Cloud**
  - is a *"Web service that provides resizable compute capacity in the cloud"*



Requirements from the Test Arch.

- Create Image
- Allocate machines, install TripCom kernel
- Configure kernel (bootstrap node?)
- Do the kernels contain initial data, or do we have to add data later (via clients → out(), before testing scalability)
- Start kernels
- Wait for them to be initialized
  - How to test that each kernel is bootstrapped and the routing tree is created? → Otherwise we have 50 instances of kernels that don't know each other...
- Start clients and their test routines
- Collect logging data from clients
- Do we need to save anything else from the kernel machines?
  - e.g. save the logs → all data on the machine instance is transient, there is no persistent storage available
- Shutdown machines so that we don’t reach Elena’s credit card limit ☺

Create Image / Allocate machine

- TripCom AMI is already created by USTUTT
- Allocate machine:
  - *ec2-run-instances ami-2bb65342 -k gsg-keypair*
    - *→ i-ae0bf0c7*
  - *ec2-describe-instances i-ae0bf0c7*
    - *“running”, name=ec2-67-202-7-236.compute-1.amazonaws.com*
  - *ec2-authorize default -p 22*
- Install stuff on running image
  - NOTE: there is no persistent storage, all stuff that is installed now will be lost after machine shutdown
  - *ssh -i id_rsa-gsg-keypair root@ec2-67-202-51-223.compute-1.amazonaws.com*
  - *scp, sftp, ...*

Configure kernel / start / validate

- *Configure kernel (optional)*
  - *ssh configure <option1> <option2> ...*
- *Start kernel*
  - *ssh startup.sh <bootstrap-node-ip> <param2> ...*
- *Validate kernel (e.g. is it registered at the bootstrap node,* ***is the routing information okay?, is the kernel ready to receive requests?*** *etc.)*
  - *ssh kernel_validate.sh → "OK"*
- Start Clients
  - ssh startup_client.sh <param1> <param2> <param3>
- Receive performance data generated by test clients
  - scp client.log
- Shutdown
  - ec2-terminate-instances instance_id,...

Prerequisites for the Tests

- Need a “bootstrap node”, and a way to tell each TripCom kernel where that node is (IP Address)
  - Either preconfigured (config file)
  - Or per command-line parameter (startup script)
  - Do we host a dedicated bootstrap node? On an external location
    - Would make testing much easier → can upload pre-configured tripcom kernels to EC2
    - Otherwise we would need to start a dedicated bootstrap node on EC2, then configure TripCom at runtime

Prerequisites for the Tests contd.

- Need an preconfigured image of the TripCom kernel
  - E.g. as .zip file with all dependencies and scripts to manage the kernel (start, stop, configure)
    - Only assumption you can make: java 1.6 is installed
    - Must be as easy as possible to deploy → need to install it on 50+ machines in an automated way. Best way possible: simply unzip and run e.g. *startup.sh <bootstrap-node-ip>*
- Need a client application that performs the tests and writes results to disk
  - Same as above: ideally as .zip file with scripts to manage lifecycle

Prerequisites for the Tests contd.

- **Need an architecture for the test cases**
  - What is the exact numbers of clients / kernels we test?
  - How many test deployments?
    - Suggestion: Max 8 (deployment takes time... and costs money ☺)
  - Do we have more than one kernel on a machine?
  - Where to host clients?
    - Suggestion: (Test-)Clients are run by the partner that conducts the actual test runs → much more flexible

Open Questions

- Who does the analysis of the data?
  - Calculate scalability measures
  - Give feedback to the development for improvements
- To conduct the final tests, we propose a small workshop where a person from every group is available and possible problems can be fixed
  - Someone from the architecture team
  - Someone from the EC2 team
  - Someone from the TripCom development Team (especially for P-GRID related problems)

TODO

- Clients running on infrastructure of the partner that executes the scalability tests
  - Thats how scalability test tools seem to work
- Proposal how to proceed:
  - Since testing on EC2 requires a lot of effort (and money) I propose
    - First (load) test the kernels and API operations on (a few) local machines
      - Kernel level scalability (distribute components) should also be tested like that
    - Only if that works good and stable under load, we can go to EC2 and test for larger scale
      - EC2 makes only sense for global operations (no URL)
      - Operations with space URL can be as well tested with a single kernel → no difference if there are 100 kernels running or just one.
  - EC2 is only used for hosting kernels: test clients are run by partners who conduct the actual scalability test work
    - → much more flexible for the partners who do the actual testing
      - Instead of calling me to re-deploy on ec2 just because something on the test clients has changed or more clients (load) are needed
    - Deployment on EC2 takes much effort an time → minimize the number of different deployments

End of Document

Instance Types

- EC2 supports different **instance types**
  - ***Small Instance***
    - 1.7 GB memory, 32-bit platform, I/O Performance: Moderate
      1 EC2 Compute Unit (1 virtual core with 1 EC2 Compute Unit)
      160 GB instance storage (150 GB plus 10 GB root partition)
      Price: $0.10 per instance hour
  - ***Large Instance***
    - 7.5 GB memory, 64-bit platform , I/O Performance: High
      4 EC2 Compute Units (2 virtual cores with 2 EC2 Compute Units each)
      850 GB instance storage (2 x 420 GB plus 10 GB root partition)
      Price: $0.40 per instance hour
  - ***Extra Large Instance***
    - 15 GB memory, 64-bit platform, I/O Performance: High
      8 EC2 Compute Units (4 virtual cores with 2 EC2 Compute Units each)
      1,690 GB instance storage (4 x 420 GB plus 10 GB root partition)
      Price: $0.80 per instance hour

Running instances

- Once an instance type has been chosen, one can define the number of instances to be created
  - 20 instance limit per user
  - After explaining what we wanted to do, our limit was extended to 50 instances
- Once an instance of an AMI is terminated, all modifications carried out after instance creations are lost
  - Run-time data that should be kept must be (manually) saved to another location (e.g. S3)