Automated Provisioning of Cloud and Cloudlet Applications

Secure and Assured Mobile Computing Components

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Jeff Boleng, PhD
May 1, 2013

Software Engineering Institute | Carnegie Mellon

DM-0000360

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

Overview

- Motivation and Goal
- Terminology
- Architectural Overview
- Surrogate Selection and Automated Provisioning
- Current Results
- Use Cases
- Next Steps and Future Work
- Questions

Team

Jeff Boleng, PhD – CMU SEI
Grace Lewis – CMU SEI
Vignesh Shenoy – CMU INI
Varun Tibrewal – CMU INI
Manoj Subramaniam – CMU INI
Sebastian Eschevarria – CMU SEI
Ben Bradshaw – CMU SEI

CMU: Carnegie Mellon University
SEI: Software Engineering Institute
INI: Information Networking Institute

Motivation and Goal

Goal

A secure and assured digital container format for mobile computing components

Technical Challenges

Myriad of languages, architectures, platforms, APIs, etc. create an exponential challenge for software portability, interoperability, security, and assurance

Key Idea

Define software portability execution characteristics and an accompanying digital container format to enable secure and assured mobile computing components

Impact

Agile software components which can be trusted and executed to provide results across the spectrum of computing platforms

Goal (expanded)

Mobile computing components that are

- Automatically provisioned
  - JIT provisioned
  - Technology agnostic
  - Dependency fulfillment
- Secure
  - Trusted computing component
  - Trusted computing provider
- Assured
  - Input and output format validation
  - Pre/post conditions
  - Run-time invariants

Approach and Assumptions

Demonstrate full end to end solution

- Each component may be sub-optimal

Digital container for computing components

- Extend DMTF OVF file format

OVF format

- Xml descriptor with packaged components (tar)

Proper packaging is critical

Ultimately demonstrate composability

Elements of the Solution

Automatically provisioned

- JIT provisioning → todays details
- Technology agnostic → run time environment specified
- Dependency fulfillment → prototype using Docker and LXC

Secure

- Trusted computing component → OVF code signing
- Trusted computing provider → Certificates and TLS

Assured

- Input and output format validation → using MIME types and Tika
- Pre/post conditions → ??? Proof carrying code ???
- Run-time invariants → ??? Proof carrying code ???

Terminology

Mobile Device

- Self explanatory

Client Application

- Application running on the mobile device communicating to the Application Server

Offload Element

- Computing component that migrates to the surrogate

Surrogate Client

- Service running on the mobile device for discovery, provisioning, and offload

Validation Service

- I/O and assurance validation

Surrogate

- Infrastructure server (cloud or cloudlet)

Surrogate Server

- Service running on the surrogate for discovery, provisioning, and offload

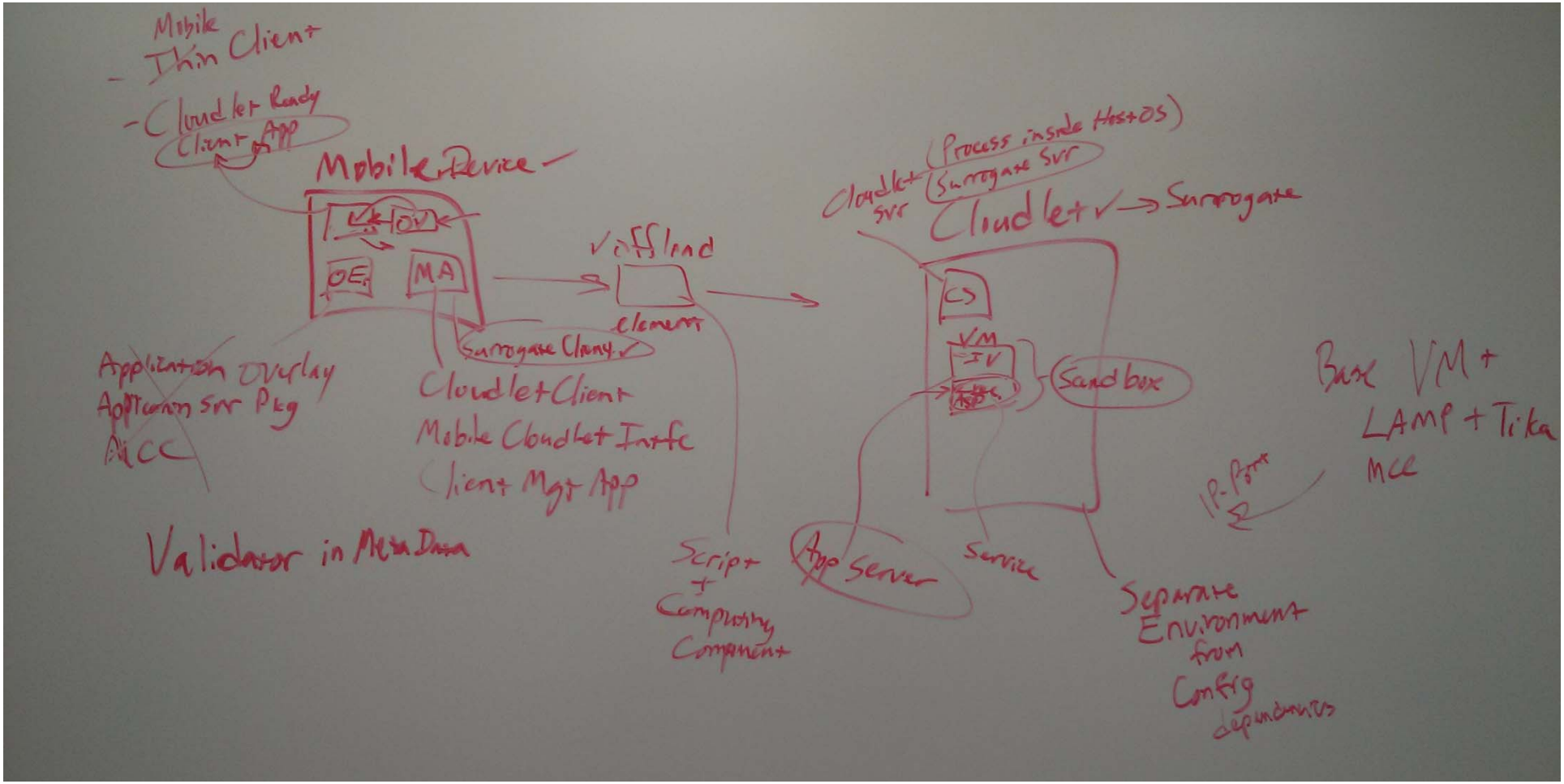
Application Server

- Run-time instantiation of the offload element

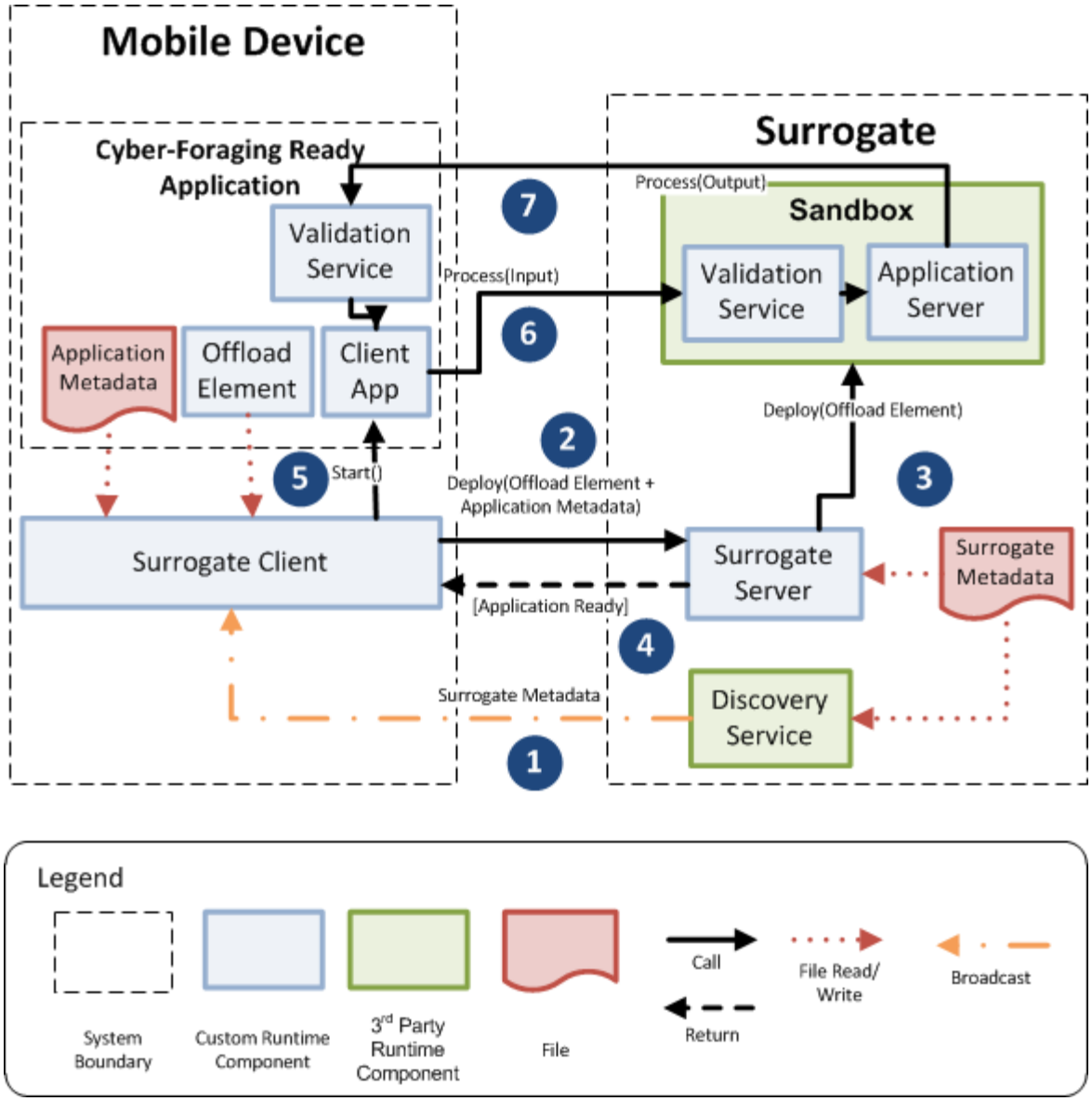
Sandbox

- Computing environment the application server executes in

Architecture Overview



Architecture Overview



1. Discovery and Provisioning

a) Surrogate Server beacons availability

b) Surrogate Client contacts Surrogate Server and establishes TLS connection

- Surrogate checks Mobile Device certificates
- Mobile Device checks Surrogate certificates

c) Surrogate Client submits requirements to Surrogate Server and keeps TLS connections alive

d) Surrog **Surrogate Selection** ogate Client

e) Surrogate Client selects best Surrogate, terminates all other TLS connections

f) Surrogate Client sends payload to selected Surrogate Server

g) Surrogate Server verifies payload integrity via code signing in metadata

h) Surrogate Server provisions an execution Sandbox for payload based on the execution type in the metadata (bytecode, script, vm, etc.)

Provisioning, Deployment, and Execution

i) Surrogate Server provisions the execution Sandbox with the Offload Element

j) Surrogate Server provisions the execution platform with Validation Service (currently using Apache Tika) and satisfies dependencies

k) Surrogate Server passes back IP address and port of Validation Service back to Surrogate Client

l) Surrogate Client starts Client Application with IP and Port

m) Client Applications sends data to Validation Service on Surrogate

n) Validation Service enforces data input format

o) Validation Service passes data to Application Service

p) Application Service executing in Sandbox on Surrogate computes result and passes it back to Validation Service on the Mobile Device

q) Validation Service on Mobile Device enforces result format

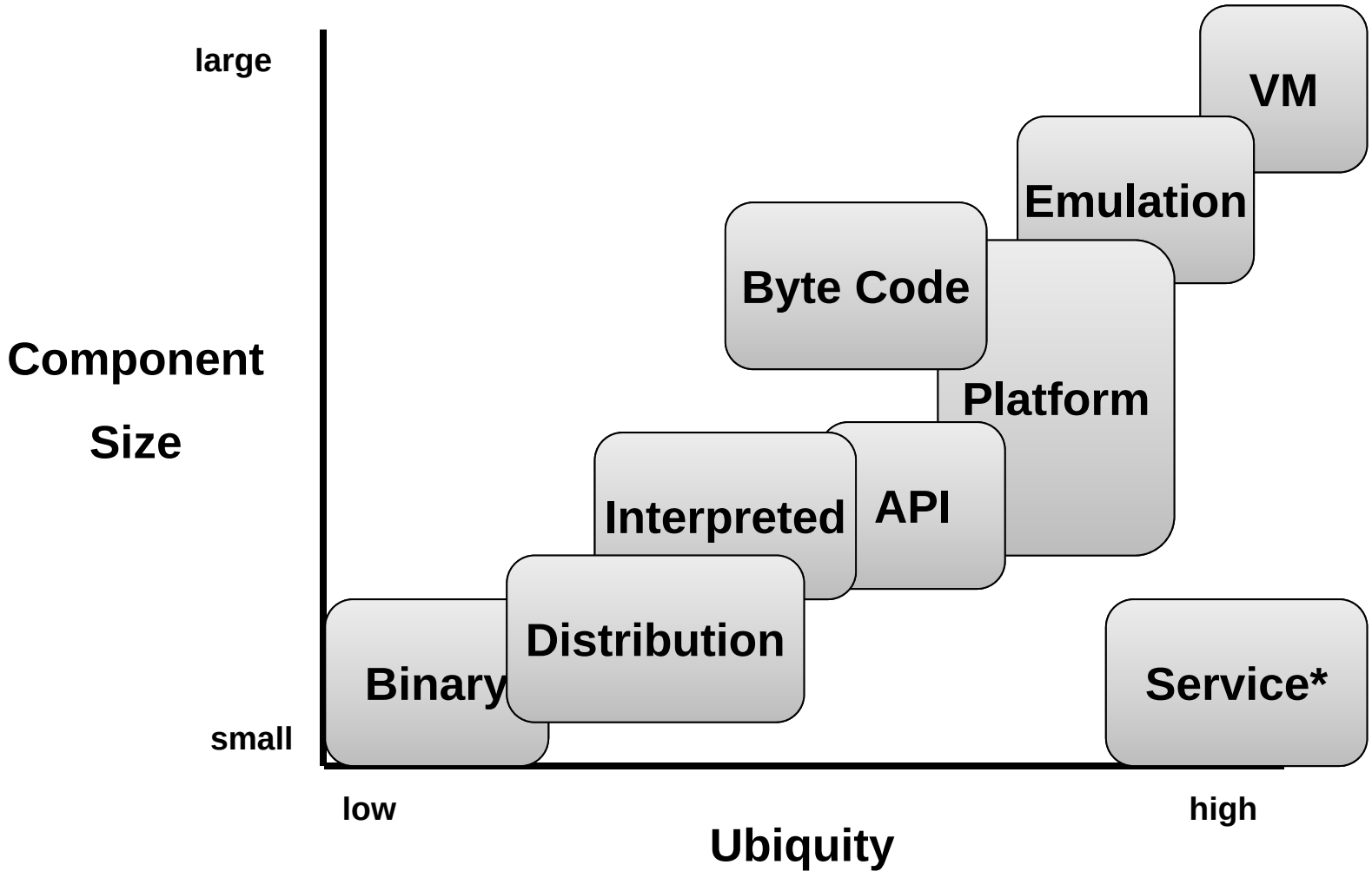
r) Validation Service passes result (or error) to Client Application

s) Loop to M or end

Run-time Environment

Software execution characteristics	
Virtual Machine	Components are portable across computing platforms and architectures using virtual machine, either with a host OS or a hypervisor
Emulation	Components are portable by running on a machine emulator
Platform	Components are portable by running in a common, standards based platform, such as a web browser (potentially enables byte code and interpreted portability)
Byte code	Portability is achieved through prior or just-in-time compilation to a common byte code and VM layer (e.g. JVM, CLR, Flash, etc.)
Interpreted	Portability is achieved with a common interpreter running on multiple computing platforms
API	Portability is achieved via source code re-compilation using a standard API (e.g. OpenGL, Qt, DirectX, etc.)
Distribution	Portability is achieved via package distribution and repositories supporting various computing platforms, either by binary distribution (e.g. rpm, apt, etc.) or source code distribution (e.g. gentoo/portage)
Binary	Binary executables run natively on different platforms with no recompilation (could require supporting OS, e.g. Portable Apps, or run as an embedded system), static vs. dynamic linkage
Service	Existing cloud service exists to provide a solution and connectivity to it is available and secure

Software execution landscape (notional)



*Requires secure and assured infrastructure access

Surrogate Selection

- Component creator determine minimum and optimal execution parameters when packaging
  - Currently memory, CPU, and disk
  - Assigns weights (1-100) that must total 100
- Multiple offload elements and multiple surrogates
- Mobile sends requirements for container execution
- Surrogate determines if it can execute component
- Surrogate calculates capability score
- Mobile calculates final performance score and selects Surrogate

Note: Investigating effects of caching

Capability Score

$$SC_i = \frac{(C_i - Ri)}{R_i}$$

$$\varphi = \sum (\mathrm{SC_i} * \mathrm{Wi})$$

SC_i – Surplus Capability of Surrogate for the resource ‘i’

C_i – Capability of surrogate for resource ‘i’ (or optimal requirement*)

R_i – Requirement of the offload element for resource ‘i’

W_i – Weight associated with resource ‘i’

φ – Capability score of the surrogate for the particular offload element

*ensures Surrogate cannot “cheat” and capture all offload elements

Size Benefit

$$sb = \frac{Max\ Size\ - Size}{Max\ Size} * 100$$

$Max\ Size$ – Size of the biggest offload element

$Size$ – Size of the offload element in consideration

Performance Score

$$\rho = \varphi + (x * sb)$$

ρ – Final performance score of the Surrogate for the Offload Element

φ – Capability score of the Surrogate for the particular Offload Element

x – Network performance factor

sb – Size benefit of the Offload Element

Testing Approach

Face detection

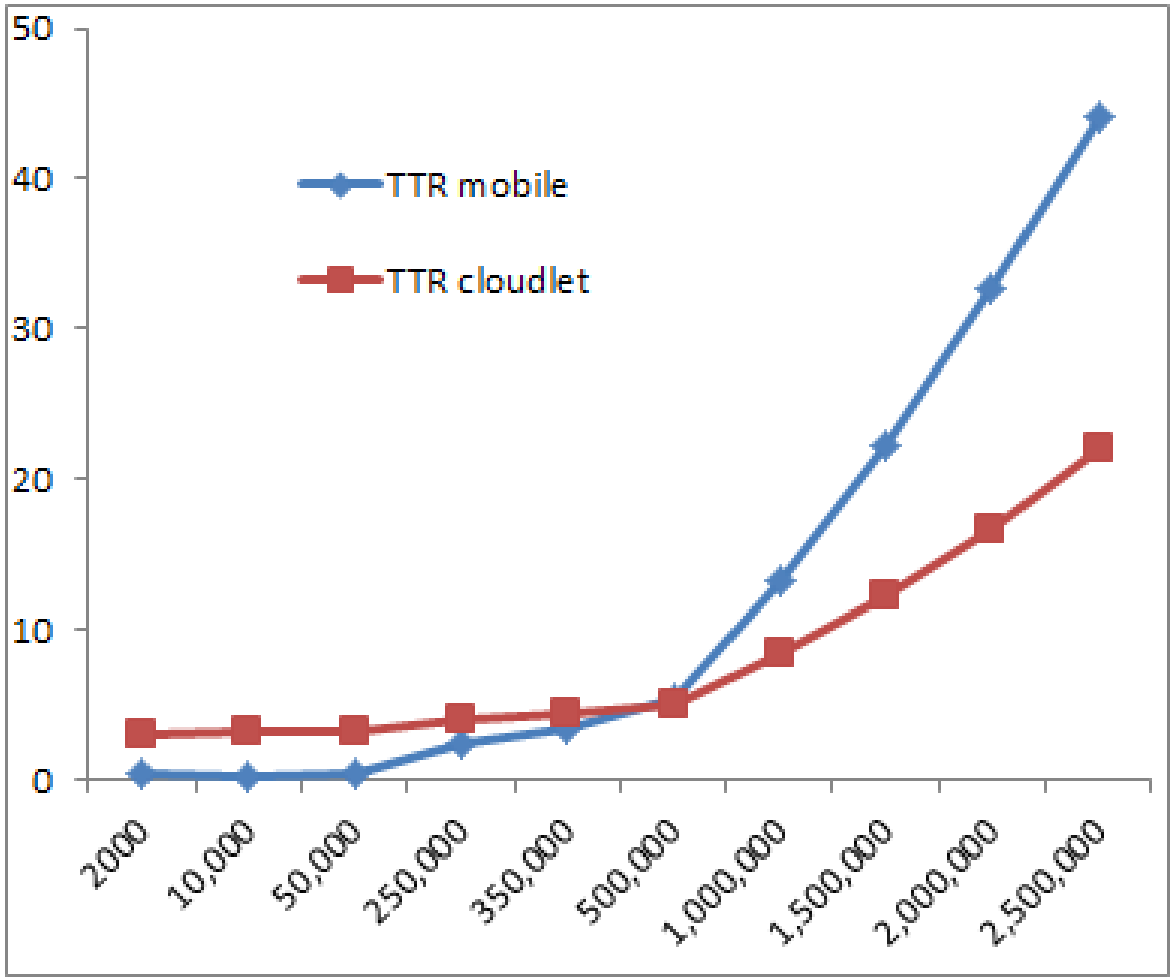
- Java and python containers

Prime number calculation

- Java, python, and C executable containers

Time-to-Result = Surrogate Discovery Time +
Surrogate SelectionTime +
Offload and Deploy time +
Input/Output communication time +
Input/Output Validation time

Sample Timing (prime number calculation)



Testing Scenario

- Select correct offload element for surrogate capability
- Select offload element with smallest weight if all capability scores were equal
- Select best surrogate with differing capability scores
- Still experimenting with TTR

Initial Test Results

RAM - min-512MB Max-2048MB

CPU - min-1GHz Max 2GHz

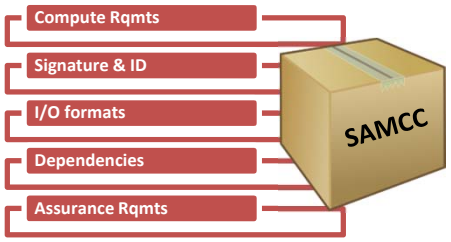
Only the weight was changed as shown

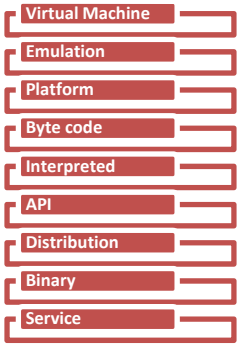
Test Case 1:	Cloudlet 1	Cloudlet 2	Choudlet 3	Selected Cloudlet	Discovery Time	Deploy Time	Average IO Time
	Java	Java	Java				
	1.0GHz, 4096MB	3.0GHz, 1000MB	2.0, 3072MB				
CPU:10,RAM:90	270	100	280	3	3179	9145	3112
CPU:20,RAM:80	240	100	260	3	3163	8920	3233
CPU:40,RAM:60	180	100	220	3	3017	10868	3305
CPU:50, RAM:50	150	100	200	3	3379	9168	4564
CPU:60, RAM:40	120	100	180	3	3024	9327	3163
CPU:80, RAM:20	60	100	140	3	3110	9012	3357
CPU:90, RAM:10	30	100	120	3			
	Java	Java	Java				
	1.0GHz, 4096MB	3.0GHz, 1024MB	2.0, 1536MB				
CPU:20,RAM:80	240	100	180	1	3329	10126	3178
CPU:40,RAM:60	180	100	160	1	3046	9467	3450
CPU:50, RAM:50	**150**	100	**150**	3	3107	9975	3450
CPU:60, RAM:40	120	100	140	3	3280	9467	3789
CPU:80, RAM:20	60	100	120	3	3100	9437	3327
CPU:90, RAM:10	30	100	110	3	3256	9045	3189

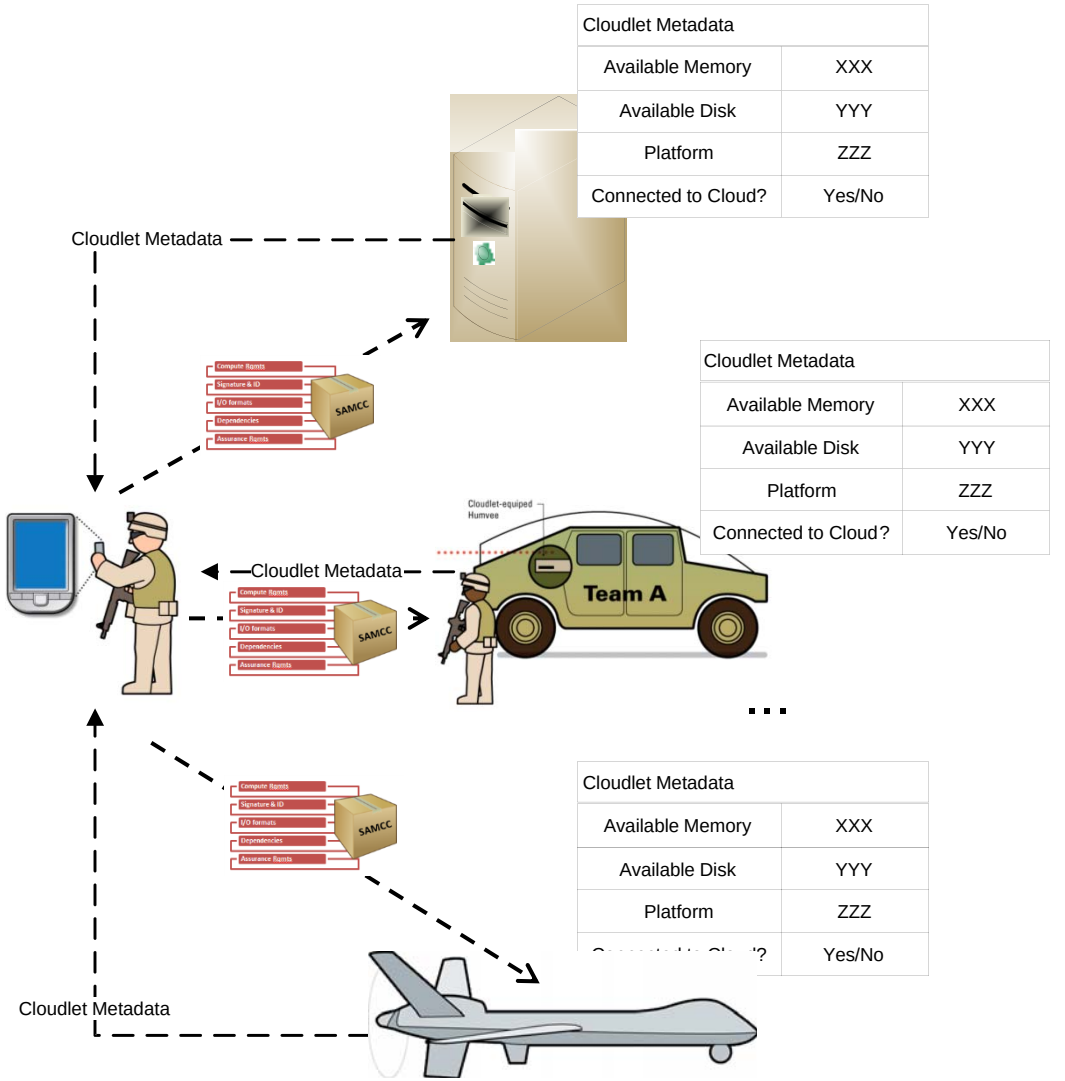
Test Case 1:	Cloudlet 1	Cloudlet 2	Choudlet 3	Selected Cloudlet	Discovery Time	Deploy Time	Average IO Time
	Java	Python	Java				
	1.0GHz, 4096MB	2.0GHz, 1536MB	2.0, 1536MB				
CPU:20,RAM:80	240, 240	180, 228	180, 180	1	3705	9787	2954
CPU:40,RAM:60	180, 180	160, 208	160, 160	2	3309	9052	3198
CPU:50, RAM:50	150, 150	150, 198	150, 150	2	3167	9783	3006
CPU:60, RAM:40	120, 120	140, 188	140, 140	2	3017	9104	3048
CPU:80, RAM:20	60, 60	120, 168	120, 120	2	3188	9227	3175
CPU:90, RAM:10	30, 30	110, 158	110, 110	2	3179	9085	3094

(Cloudlet score, final perfomance score considering the size of the package)

Use Cases







Use Cases

- Cyber foraging
- Rapid re-configuration of deployed sensors
- Distribute search logic in large scale distributed databases
- Interoperability or translation for existing or legacy systems
- Service market place of composable computing components
  - Image processing
  - Speech recognition
- Crowd analysis
  - Extend MIT “mood meter” with offload elements and mobile devices
  - POI location via face detection and recognition

Next Steps and Future Work

- Finish prototype using Docker and all other existing components
- More testing
  - Dependency satisfaction
  - Input/Output validation
    - Need to get output validation functioning on mobile device
  - Wide range of offload elements
  - Wide configuration of Surrogates
- Open source the solution
- Propose formal extensions to DMTF OVF (tentative)
- Demonstrate component composability
- Examine deeper assurance solutions (e.g. DNS Server component)
  - Pre/post condition specification and enforcement
  - Run-time invariant specification and enforcement
- Populate project artifacts on SATURN blog

Questions and Feedback

Thank you for your time and attention.

We welcome feedback and collaboration.

Please contact me at jlboleng@sei.cmu.edu with questions or feedback.

http://saturnnetwork.wordpress.com/category/secure-and-assured-mobile-computing-components/