Titanium Server

Resource Scaling

Host-Guest Message-Based API
&
Guest Reference Implementation

Release: 15.12

01/14/2016

WIND RIVER

[Specification of the Host-Guest Resource Scaling APIs – 16.01.]

Copyright Notice

Table of Contents

Introduction

Titanium Server implements a Host-to-Guest Resource Scaling Messaging-based API to co-ordinate resource scaling operations between the Titanium Server Host Platform and the Guest VM. Currently scaling the number of online vCPUs within the Guest is supported. This document contains the specification for this messaging-based API.

Also included in this document is an overview of the Titanium Server Guest Resource Scaling SDK Module which provides a Linux-based reference implementation of the Guest-side software for implementing this Messaging-based API in the Guest and implementing the appropriate management of the scaled resources in the Guest. I.e. in the case of resource scaling of CPUs, this involves the appropriate on-lining and off-lining of CPUs inside the Guest VM. The SDK Module provides source code and make/build instructions which can be used strictly as reference or built and included ‘as is’ in your Guest image. Full build, install and usage instructions can be found in the README files included in the SDK Module. This document simply provides an overview of the reference implementation.

Host – Guest Resource Scaling API

Titanium Server implements a simple Host-to-Guest Resource Scaling API to co-ordinate resource scaling operations between the Titanium Server Host Platform and the Guest VM. Currently scaling the number of online vCPUs within the Guest is supported.

The Host-to-Guest Resource Scaling API is a message-based API using a JSON-formatted application messaging layer on top of a 'virtio serial device' between QEMU on the host and the Guest VM. JSON formatting provides a simple, humanly readable messaging format which can be easily parsed and formatted using any high level programming language being used in the Guest VM (e.g. C, Python, Java, etc.). Use of the 'virtio serial device' provides a simple, direct communication channel between host and guest which is independent of the Guest's L2/L3 networking.

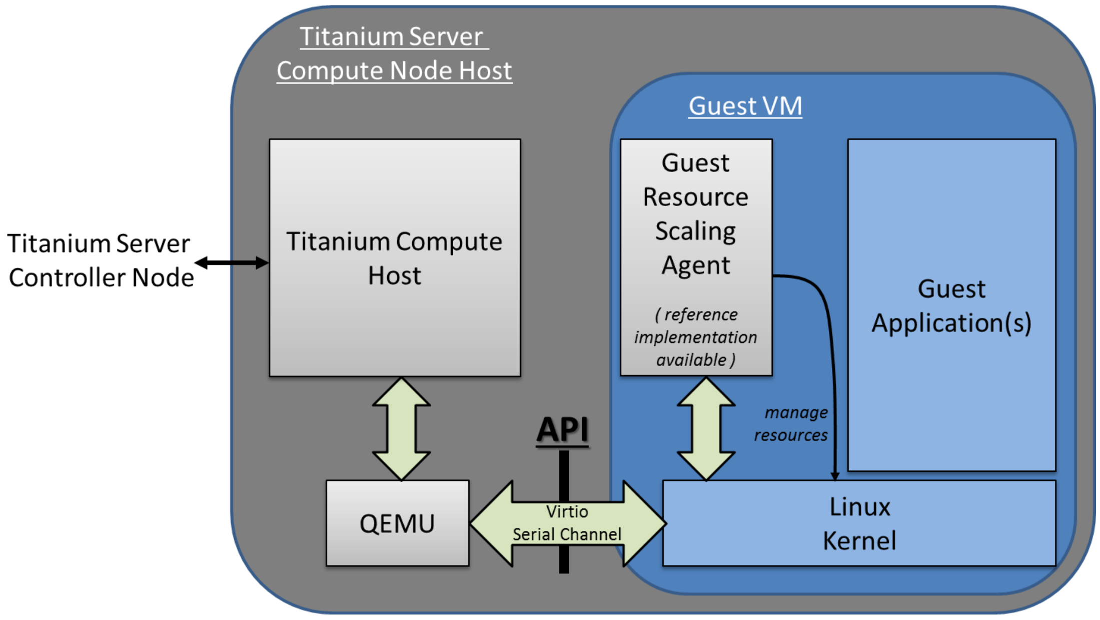
Titanium Server Compute Node Host
Guest VM
Titanium Server Controller Node
Titanium Compute Host
Guest Resource Scaling Agent
(reference implementation available)
Guest Application(s)
manage resources
API
QEMU
Virtio Serial Channel
Linux Kernel

Figure 1 – Titanium Server Host-Guest Resource Scaling API

Message Types and Semantics

For the Resource Scaling API, there are only three message types; a Resource Scaling Request from the Host to the Guest, a Resource Scaling Response from the Guest to the Host and a Nack Message.

- Resource Scaling Request (Host → Guest)
  - This is the Resource Scaling Request sent from Titanium Server to the Guest VM.

    It specifies the resource to be scaled (currently CPU scaling, but memory and other resources may be added in the future) and the direction of scaling (i.e. up or down). In the case of scaling CPU up, the specific CPU number to be on-lined and the expected resulting full list of on-lined CPUs (after the scale up completes) is specified as well.
  - On receiving a CPU Scale Up request, the Guest is expected to:
    - on-line the specified CPU, using the mechanisms appropriate to the Guest's OS,
      - for example, on Linux, using the /sys/devices/system/cpu virtual filesystem to read and modify the online status of individual cores of the cpu device(s).
    - respond to the request with the 'Resource Scaling Response' as described below.
  - On receiving a CPU Scale Down request, the Guest is expected to:
    - off-line the specified CPU, using the mechanisms appropriate to the Guest's OS,
    - respond to the request with the 'Resource Scaling Response' as described below.

- Resource Scaling Response (Guest → Host)
  - This is the Resource Scaling Response sent from the Guest VM to the Titanium Server in response to the Resource Scaling Request.

    It echoes back the resource being scaled and the direction of scaling. It also specifies the result of the Guest Scaling Operation (i.e. success or fail), and for the fail scenario also specifies the details of the error which occurred.

    In addition, for a successful scale up, the CPU number on-lined is echoed back as well as the resulting list of online CPUs. And similarly for a successful scale down, the off-lined CPU number is returned, and the resulting list of online CPUs.

- Nack (Host → Guest)
  - This is a message sent from the Host to the Guest when the Host receives a message with incorrect syntax,
  - It contains the message type of the original (incorrect) message and a log_msg describing the error,
  - This allows the Guest Application developer to debug issues when developing the Guest-side API code.

Virtio Serial Device

The transport layer of the Host-Guest Resource Scaling API is a ‘virtio serial device’ (also known as a ‘vmchannel’) between QEMU (on the host) and the Guest VM. Device emulation in QEMU presents a virtio-pci device to the Guest, and a Guest Driver presents a char device interface to Guest userspace applications. This provides a simple transport mechanism for communication between the host userspace and the guest userspace. I.e. it is completely independent of the networking stack of the Guest, and is available very early in the boot sequence of the Guest.

This is a standard Linux QEMU/KVM feature. The Guest API for interfacing with the ‘virtio serial device’ can be found at http://www.linux-kvm.org/page/Virtio-serial_API . Examples of Guest code for opening, reading, writing, etc. from/to a ‘virtio serial device’ can also be found in the source code of the Titanium Server Guest Resource Scaling SDK Module. This SDK Module provides a Linux-based reference implementation of the Guest-side software for implementing the Guest Resource Scaling Messaging API and implementing the requested scaling of Guest Resources. Generally communicating with a ‘virtio serial device’ is very similar to communicating via a pipe, or a SOCK_STREAM socket.

There are however a few additional considerations to be aware of when using ‘virtio serial devices’:

- only one process at a time can open the device in the Guest,
- read() returns 0, if the Host is not connected to the device,
- write() blocks or returns -1 with error set to EAGAIN, if the Host is not connected,
- poll() will always set POLLHUP in revents when the Host connection is down.
    - This means that the only way to get event-driven notification of connection is to register for SIGIO. However, then a SIGIO event will occur every time the device becomes readable. The work-around is to selectively block SIGIO as long as the link is up is thought to be up, then unblock it on connection loss so a notification occurs when the link comes back.
- If the Host disconnects the Guest should still process any buffered messages from the device,
- Message boundaries are **not preserved**, the Guest needs to handle message fragment reassembly. Multiple messages can be returned in one read() call, as well as buffers beginning and ending with partial messages. This is hard to get perfect; one can study the `host_guest_msg.c` code in the Titanium Server Guest Resource Scaling SDK Module for ideas on how this can be handled.

The QEMU/KVM created by Titanium Server in order to host a Guest VM is created with a 'virtio serial device' named:

```
      /dev/virtio-port/cgcs.messaging
```

for general Titanium Server Host – to – Guest VM messaging (e.g. Host-Guest Resource Scaling Messaging as well as other Host-Guest Messaging discussed in other Titanium Server – Guest API documents).

JSON Message Syntax

The upper layer messaging format being used is ‘Line Delimited JSON Format’. I.e. a ‘\n’ character is used to identify message boundaries in the stream of data to/from the virtio serial device; specifically a ‘\n’ character is inserted at the start and end of the JSON Object representing a Message.

```
\n{key:value,key:value,…}\n
```

Note that key and values must NOT contain ‘\n’ characters.

The upper layer messaging format is actually structured as a hierarchical JSON format containing a Base JSON Message Layer and an Application JSON Message Layer:

- the Base Layer provides the ability to multiplex different groups of message types on top of a single ‘virtio serial device’
  e.g.
  - resource scaling,
  - server group messaging,
  - etc.

  and
- the Application Layer provides the specific message types and fields of a particular group of message types.

Base JSON Message Layer – Syntax

Again, the Base Layer provides the ability to multiplex different groups of message types on top of a single ‘virtio serial device’, e.g. resource scaling versus server group messaging etc.

Host – to – Guest Messages

Key	Value	Optionality*	Example value *(for Resource Scaling)*	Description
“version”	integer	M	1	Version of the Base Layer Messaging
“source_addr”	string	M		Opaque string representing the host-side address of the message.
“dest_addr”	string	M	“cgcs.scale”	The Guest-side addressing of the message; specifically the Message Group Type
“data”	JSON Formatted String	M	See the following section on Application Layer JSON Message Layer – Syntax for Resource Scaling.	Application layer JSON message whose schema is dependent on the particular Message Group Type

- M: Mandatory; O: Optional

Guest – to – Host Messages

Guest – to – Host Messages, from a Base Layer perspective, are identical to Host – to – Guest Messages except for swapped semantics of source_addr and dest_addr.

Application JSON Message Layer – Syntax

Again the Application Layer provides the specific message types and fields of a particular group of message types; in this case the messages of Resource Scaling.

<u>Host – to – Guest Messages</u>

<u>Resource Scaling Request</u>

Key	Value	Optionality*	Example value	Description
“version”	integer	M	1	Version of the Application Layer Messaging
“msg_type”	string “scale_request”	M	“scale_request”	Type of message
“timeout_ms”	integer	M	500	Timeout in milli-seconds to be used for the internal Guest operation to scale the resource.
“resource”	string “cpu”	M	“cpu”	The resource to scale. CPU scaling is currently supported.
“direction”	string “up” or “down”	M	“up”	Direction of scaling
“online_cpu”	integer	M(direction=up)	4	vCPU number to online when scaling up.
“online_cpus”	array of integers	M(direction=up)	[0,1,2,3,4]	The expected array of current online cpus after the command completes, if successful.

- M: Mandatory; O: Optional; (Condition)

<u>Nack</u>

Key	Value	Optionality*	Example value	Description
“version”	integer	M	1	Version of the interface
“msg_type”	string	M	“scale_response_nack”	Type of message
“log_msg”	string	M	“failed to parse version”	Error message

- M: Mandatory; O: Optional; (Condition)

WIND™

Guest – to – Host Messages

Resource Scaling Response

Key	Value	Optionality*	Example value	Description
“version”	integer	M	1	Version of the Application Layer Messaging
“resource”	string “cpu”	M	“cpu”	The resource to scale. CPU scaling is currently supported.
“direction”	string “up” or “down”	M	“up”	Direction of scaling
“result”	string “success” or “fail”	M	“fail”	Result of the scaling operation
“online_cpu”	integer	M(direction=up, result=success)	4	vCPU number which was onlined, if scaling up and successful
“offline_cpu”	integer	M(direction=down, result=success)	5	vCPU number which was offlined, if scaling down and successful
“online_cpus”	array of integers	M(result=success)	[0,1,2,3,4]	Array of current online cpus when sending response.
“err_msg”	string	M(result=fail)	“failed to scale up”	Error message.

- M: Mandatory; O: Optional; (Condition)

WIND™

Examples

Examples of ‘full’ Resource Scaling JSON messages, containing the Application JSON Message Layer encapsulated inside the Base JSON Messaging Layer.

Scale Up Request:

TiS’ request to scale up by onlining CPU#4 (Current online CPUs are [0,1,2,3].):

```
\n{"version":1,"source_addr":"scale-########”,"dest_addr":"cgcs.scale”,
"data":{"version":1,"timeout_ms":500,"resource":"cpu","direction":"up","onlin
e_cpu":4,"online_cpus":[0,1,2,3,4]}}\n
```

Guest’s Success Response to Scale Up:

```
\n {"version":1,"source_addr":"cgcs.scale”,"dest_addr":"scale-
########”,"data":{"version":1,"resource":"cpu","direction":"up","online
_cpu":4,"result":"success","online_cpus":[0,1,2,3,4]}} \n
```

Guest’s Failed Response to Scale Up:
i.e. indicating an error due to a guest that only supports 4 CPUs:

```
\n {"version":1,"source_addr":"cgcs.scale”,"dest_addr":"scale-
########”,"data":{"version":1,"resource":"cpu","direction":"up","online
_cpu":4,"result":"fail","err_msg":"Can't open cpu online path:
/sys/devices/system/cpu/cpu4/online"}}\n
```

Scale Down Request:

TiS’ request to scale down (Current online CPUs are [0,1,2,3,4,5]):

```
\n {"version":1,"source_addr":"scale-########”,"dest_addr":"cgcs.scale”,
"data":{"version":1,"timeout_ms":500,"resource":"cpu","direction":"down"}}\n
```

Guest’s Success Response to Scale Down:

```
\n {"version":1,"source_addr":"cgcs.scale”,"dest_addr":"scale-
########”,"data":{"version":1,"resource":"cpu","direction":"down","resu
lt":"success","offline_cpu":5,"online_cpus":[0,1,2,3,4]}}\n
```

Guest’s Failed Response to Scale Down:

```
\n {"version":1,"source_addr":"cgcs.scale”,"dest_addr":"scale-
########”,"data":{"version":1,"resource":"cpu","direction":"down","resu
lt":"fail","err_msg":"Some failed message"}}\n
```

Nack:

A Nack from TiS for an invalid response message sent from Guest.

```
\n{"version":1,"source_addr":"cgcs.scale”,"dest_addr":"cgcs.scale”,"data":{"v
ersion":1,"msg_type":"nack","log_msg":"failed to parse version"}}\n
```

Reference Implementation of Guest Resource Scaling

This section provides an overview of the Linux-based reference implementation of the Guest-side software for implementing this Host-to-Guest Resource Scaling Messaging-based API in the Guest and implementing the appropriate management of the scaled resources in the Guest. I.e. in the case of resource scaling of CPUs, this involves the appropriate on-lining and off-lining of CPUs inside the Guest VM.

This reference implementation can be found in the Titanium Server Guest Resource Scaling SDK Module. This Module provides source code and make/build instructions which can be used strictly as reference or built and included 'as is' in your Guest image. Full build, install and usage instructions can be found in the README files included in the SDK Module. This section simply provides an overview of the reference implementation.

The diagram below provides the architecture diagram of the reference implementation:

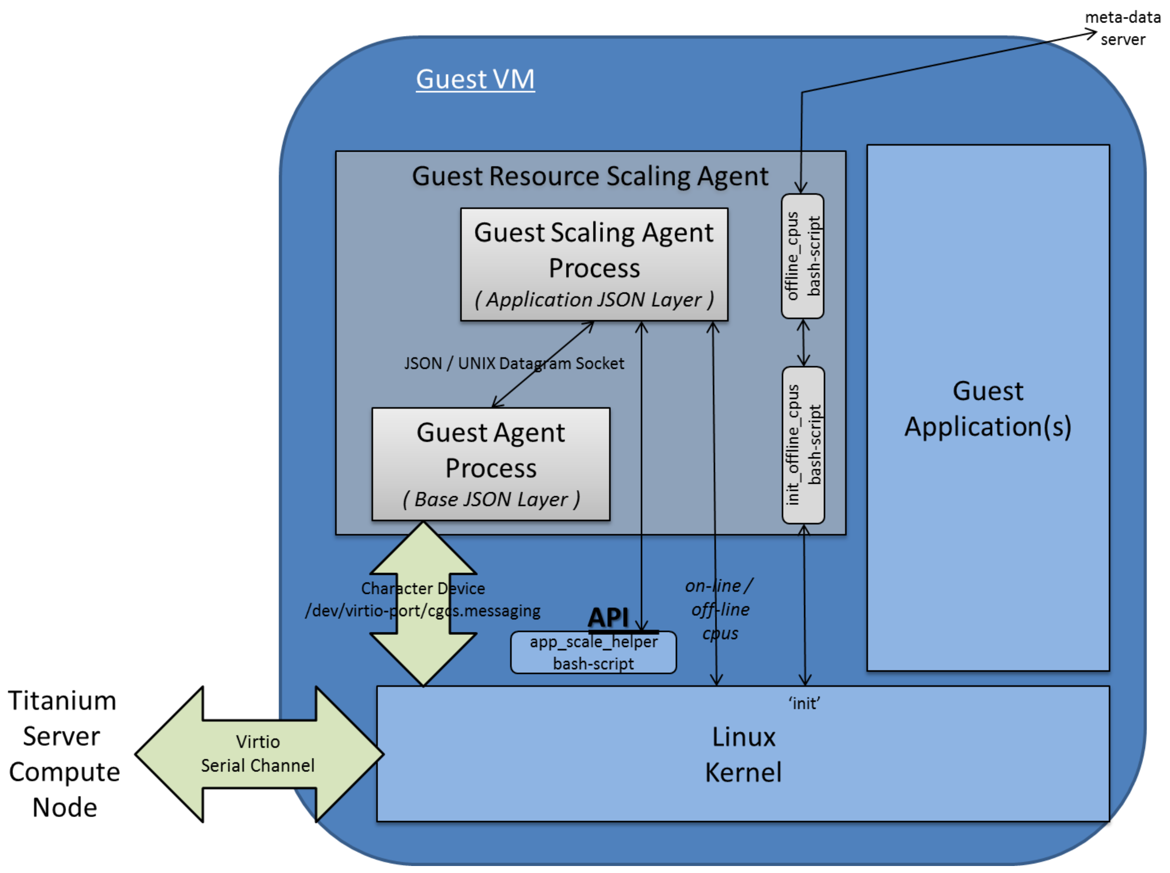
meta-data server
Guest VM
Guest Resource Scaling Agent
Guest Scaling Agent Process
(Application JSON Layer)
offline_cpus bash-script
JSON / UNIX Datagram Socket
init_offline_cpus bash-script
Guest Agent Process
(Base JSON Layer)
Guest Application(s)
Character Device /dev/virtio-port/cgcs.messaging
API
app_scale_helper bash-script
on-line / off-line cpus
'init'
Linux Kernel
Titanium Server Compute Node
Virtio Serial Channel

Figure 2 – Reference Implementation Architecture for Guest Resource Scaling

Where:

- A Guest Agent Process implements the Base JSON Messaging Layer.
  This includes:
  - opening/reading,/writing and general management of the virtio serial device between the Guest and the Host,
  - parsing/processing/formatting of the Base JSON Messaging Layer of the Guest-Host interface, where processing of the messages involves:
    - the multiplexing/de-multiplexing of Application Layer messages to/from registered Guest Application Layer Agents; in this particular case the Guest Resource Scaling Agent,
    - the interface between the Guest Agent Process and the Guest Resource Scaling Agent process
      - is a message-based interface;
      - specifically a JSON Messaging Layer over a UNIX Datagram socket,
        - where the UNIX Socket Address is the Message Group Type (cgcs.scaling in this particular case) specified within the Base JSON Messaging Layer and
        - where the JSON Message consists of the 'data' field contents specified within the Base JSON Messaging Layer.
  - NOTE
    - This Guest Agent Process reference implementation is actually contained in the Titanium Server Guest Server Group Messaging SDK Module, and used to implement the Base JSON Messaging Layer for both Server Group Messaging and Resource Scaling Messaging
    - The implementation files are:
      - misc.h, guest_host_msg.h, host_guest_msg_type.h,
      - guest_agent.c, host_guest_msg.c, lib_guest_host_msg.c

- A Guest Scaling Agent Process implements the Resource Scaling Application JSON Messaging Layer and the code for management of the scaled resources in the Guest. I.e. the appropriate on-lining and off-lining of CPUs inside the Guest VM.

  As described above
  - the interface between the Guest Agent Process and the Guest Resource Scaling Agent process is a JSON Messaging Layer over a UNIX Datagram socket.
    - where the UNIX Socket Address is the Message Group Type (cgcs.scaling in this particular case) specified within the Base JSON Messaging Layer and
    - the JSON Message consists of the 'data' field contents specified within the Base JSON Messaging Layer.

- the on-lining and off-lining of CPUs is done by using the /sys/devices/system/cpu virtual filesystem to read and modify the online status of individual cores of the cpu device(s),

    - <u>/usr/sbin/app_scale_helper SCRIPT-BASED API</u>

      as part of the on-lining and off-lining CPU procedure,
      the /usr/sbin/app_scale_helper script is called in in order to provide a script-based mechanism for the Guest Application to modify the default behavior.

      Specifically:

        - before off-lining the CPU, Guest Scaling Agent will make a call to

          /usr/sbin/app_scale_helper --cpu_del

          (if it exists) which is expected to pick a vCPU to offline by returning the selected vCPU number.

        - after on-lining the CPU, Guest Scaling Agent will make a call to

          /usr/sbin/app_scale_helper --cpu_add <cpu> <new cpu range>

          (if it exists) passing the vCPU number on-lined so the application can do any special handling that may be required (e.g. modifying the affinity of processes to leverage the new vCPU).

- and finally, the Guest Scaling Agent implementation provides an init_offline_cpus and offline_cpus script to offline vCPUs in the Guest to match the status on the hypervisor. This covers the case where the Guest VM is booting up with some CPUs offlined by the hypervisor.

- NOTE
    - Guest Scaling Agent reference implementation is contained in the Titanium Server Guest Resource Scaling SDK Module.
      The implementation files are:
        - misc.h
        - parser.c, guest_scale_agent.c
        - app_scale_helper, init_offline_cpus, offline_cpus