Resource Management with systemd

LinuxCon North America 2013

Lennart Poettering

September 2013

Resource Management?

Distributing available CPU, IO, and memory resources between services/applications

Distributing available CPU, IO, and memory resources between services/applications

On embedded: limited resources, lots of things to run

Distributing available CPU, IO, and memory resources between services/applications

On embedded: limited resources, lots of things to run

On servers: a lot of resources, maximization of density

Distributing available CPU, IO, and memory resources between services/applications

On embedded: limited resources, lots of things to run

On servers: a lot of resources, maximization of density

Underlying technology for systemd: Linux kernel control groups

Control Groups

Control Groups

First used by systemd merely for grouping processes

Control Groups

First used by systemd merely for grouping processes

Original purpose from the kernel side though: resource management

systemd hides the fact that cgroups is used underneath

systemd hides the fact that cgroups is used underneath

cgroups are now an implementation detail

(Can I still use cgroups without systemd?)

(Can I still use cgroups without systemd?)

(Why is this a job for systemd?)

systemd's resource management is based on units

systemd's resource management is based on units

Services, Scopes, Slices

Service = A group of processes, which systemd started based on unit configuration. (Example: apache.service)

Service = A group of processes, which systemd started based on unit configuration. (Example: apache.service)

Scope = A group of processes, which others have started and registered using runtime APIs (Example: fedora17.scope)

Service = A group of processes, which systemd started based on unit configuration. (Example: apache.service)

Scope = A group of processes, which others have started and registered using runtime APIs (Example: fedora17.scope)

Slice = A unit to build a hierarchy to place service and scope units in (Example: customer1.slice)

Service = A group of processes, which systemd started based on unit configuration. (Example: apache.service)

Scope = A group of processes, which others have started and registered using runtime APIs (Example: fedora17.scope)

Slice = A unit to build a hierarchy to place service and scope units in (Example: customer1.slice)

(User sessions, containers, VMs are exposed as scopes.)

Service = A group of processes, which systemd started based on unit configuration. (Example: apache.service)

Scope = A group of processes, which others have started and registered using runtime APIs (Example: fedora17.scope)

Slice = A unit to build a hierarchy to place service and scope units in (Example: customer1.slice)

(User sessions, containers, VMs are exposed as scopes.)

Slices do not contain process, they simply organize a hierarchy in which scopes and services may be placed, which in turn contain the processes

Slices are organized in a hierarchy, the name of a slice unit corresponds with the path to the location in the hierarchy.

Slices are organized in a hierarchy, the name of a slice unit corresponds with the path to the location in the hierarchy.

Examples:

foo.slice, foo-bar.slice

customer1.slice, customer1-departmentA.slice, customer1-departmentA-projectalpha.slice

Slices are organized in a hierarchy, the name of a slice unit corresponds with the path to the location in the hierarchy.

Examples:

foo.slice, foo-bar.slice

customer1.slice, customer1-departmentA.slice, customer1-departmentA-projectalpha.slice

systemd-cgls is your friend!

Default:

```
+ system.slice
| + systemd-udevd.service
| + systemd-logind.service
| + systemd-journald.service
| + apache.service
| + mysql.service
+ user.slice
|  + user-100.slice
|    + session-1.scope
+ machine.slice
  + fedora-20.scope
```

Example:

```
+ customer1.slice
| + customer1-apache.service
| + customer1-mariadb.service
+ customer2.slice
  + customer2-departmentA.slice
  | + customer2-departmentA-apache.service
  | + customer2-departmentA-mariadb.service
  + customer2-departmentB.slice
    + customer2-departmentA-postgresql.service
    + customer2-departmentA-rhel7.scope
    + customer2-departmentA-rhel6.scope
```

Every user automatically gets his own slice when he logs in

Every user automatically gets his own slice when he logs in

Every user session automatically gets its own scope within that slice

Every user automatically gets his own slice when he logs in

Every user session automatically gets its own scope within that slice

Every templated service automatically gets a slice for grouping all instances

Example:

```
+ customer1.slice
  + customer1-apache.slice
    + apache@website1.service
    + apache@website2.service
```

Arranging units in slices

Arranging units in slices

Slice=

Setting resources on units

Setting resources on units

CPUAccounting=1, CPUShares=

Setting resources on units

CPUAccounting=1, CPUShares=

MemoryAccounting=1, MemoryLimit=, MemorySoftLimit=

Setting resources on units

CPUAccounting=1, CPUShares=

MemoryAccounting=1, MemoryLimit=, MemorySoftLimit=

BlockIOAccounting=1, BlockIOWeight=, BlockIODeviceWeight=,
BlockIOReadBandwidth=, BlockIOWriteBandwidth=

Setting resources on units

CPUAccounting=1, CPUShares=

MemoryAccounting=1, MemoryLimit=, MemorySoftLimit=

BlockIOAccounting=1, BlockIOWeight=, BlockIODeviceWeight=,
BlockIOReadBandwidth=, BlockIOWriteBandwidth=

DeviceAllow=, DevicePolicy=

For services and slices in unit files or drop-ins:

For services and slices in unit files or drop-ins:

```
[Unit]
Description=Foobar Daemon

[Service]
ExecStart=/usr/bin/foobard
CPUShares=600
MemoryLimit=500M
```

At runtime with systemctl:

At runtime with systemctl:

```
$ systemctl set-property httpd.service CPUShares=600
MemoryLimit=500M
```

. . . from your app via bus calls

Monitoring

Monitoring

systemd-cgtop

Monitoring

systemd-cgtop

Don't forget to enable CPU/Memory/BlockIO accounting!

There's more to resource management!

Nice=, IOSchedulingClass=, IOSchedulingPriority=,
CPUSchedulingPolicy=, CPUSchedulingPriority=, CPUAffinity=,
TimerSlackNS=, LimitCPU=, ...,

There's more to resource management!

Nice=, IOSchedulingClass=, IOSchedulingPriority=,
CPUSchedulingPolicy=, CPUSchedulingPriority=, CPUAffinity=,
TimerSlackNS=, LimitCPU=, ...,

Not dynamically changable for units

That's all, folks!