Advanced Blade Capability Advertisement

Service Key Introduction

Tractor matches job requirements to blade capabilities using a simple abstract keyword matching system called **service keys.**

In a job, requirements are specified with the RemoteCmd -service "keys" statement, where *"keys"* can be a simple keyword, or a more complex expression.

In blade profiles, capabilities are enumerated in the *Provides <tractor_profiles.html#provides-svckeys>* list. The names in the list represent attributes of those blades: they are running a particular OS, or have some software package installed, or have fast network connections, etc. The key names are arbitrary and only have a site-defined meaning in as much as they match the -service names placed into jobs that require those blade attributes.

So a **blade profile** (in blade.config) might contain this entry:

```
"Provides": ["PixarRender", "RfMRender", "Linux"]
```

and a **job** might have commands like:

```
RemoteCmd {prman test.rib} -service "RfMRender"
RemoteCmd {prman test.rib} -service "PixarRender && Linux"
```

Advanced Capability Annotations

Several advanced blade profile *"Provides"* modes have been added to support some unusual usage cases. Note that these annotations go in *blade.config* **not** in job scripts!

Counted Service Keys

The **(max:NN)** suffix.

It can sometimes be useful to treat the "Provides" list as a set of counted resources on each blade host. Tractor keys can be annotated to indicate that their usage should be tracked by the blade and that the blade should only advertise service availability for those keys that still have unused capacity.

Note that this "counted key" feature is subtle and typically only useful in some unusual cases. It overlaps with both regular **limits**processing and also the generic blade **slot count** concept. The counting aspect is only meaningful when a single blade host is allowed to run more than one command concurrently; that is: when the blade's *slot count* is greater than one.

The idea is to "allocate" a fixed maximum concurrency count to each service key type on a blade. Sometimes this need arises when commands using a particular key are known to have a large resource usage "footprint" while programs using other keys are lightweight. For example, a studio may have typical prman jobs that are resource-intensive, but also have simple nuke jobs that are lightweight.

```
"Provides": ["PixarRender(max:2)", "NukeRender(max:4)", "Linux"],
"Slots": 4,
...
```

In this example, blades using this profile begin by accepting work from any job requiring the PixarRender or NukeRender service. They have four total concurrency slots. At any given time there can be no more than two PixerRender processes and four NukeRender processes. As a result the blade may be 1 prman and 3 nukes, or two prman and 2 nukes, or 4 nukes.

So this feature allows blades to dynamically advertise a different set of service key capabilities depending on which keys have already been "consumed" by previously launched commands.

NOTE: these counted service keys are tracking *blade capabilities* which are conceptually independent of global Limit counts and resource sharing allocations at the *project or licensing* level. These counted keys determine whether a given **blade has remaining capacity** for work, whereas Limits are applied when determining which jobs, if any, can be assigned to those available blades.

Contingent Service Advertisement

The **(after:KEY)** suffix.

This blade profile "Provides" key annotation is related to the Counted Service Keys concept described above. In this variation, the slots on a blade are directed to be assigned to only one kind of job before allowing other "filler" types to use up possible remaining capacity. An example might be that a group of many-core machines should each be reserved for two 8-thread prman invocations, with remaining capacity to be filled by nuke jobs. The goal is to always leave a large prman opening available when idle, rather than allowing lightweight jobs to nibble away at the available slots never leaving a large enough vacancy for the prman task to get scheduled.

```
"Provides": ["PixarRender(max:2)", "NukeRender(after:PixarRender)"],
"Slots": 6,
...
```

In this example the NukeRender service capability will not even be advertised until two PixarRender processes are running. The "after" annotation indicates that the given key is only available after the named key has reached its maximum count. If one of the running prman processes exits, then PixarRender will fall below its max 2 and no new nuke process will be assigned until another prman lands. A typical assumption in these cases is that the "after" key is associated with lightweight, fast-running programs.

Again, this feature is used to control whether a blade capability is considered available at all based on currently running commands. It is independent of Limits but can affect the total count of running programs of a particular type.

Required Keys: Enforcing Mandatory Job Service Types

The **(R)** suffix.

Adding the optional "keyname(R)" suffix to names in profile "Provides" list changes the usual blade assignment logic in a subtle way. The key becomes *ma ndatory,* jobs are *required* to ask for that key amongst others to match the blade.

The usual handling of names in a blde "Provides" list is that a match to *ANY* key in the list is sufficient to allow the matching command to run on that blade. The "(R)" suffix specifies instead that commands are **REQUIRED** to request at least the given key in order to use that blade.

Here is an example Provides list using the annotation:

```
    "Provides": ["PixarRender", "RfMRender", "DebugEnv(R)"]
```

Given blades using that profile, then RemoteCmds using "-service PixarRender" alone would *no longer* run on those blades, but "-service PixarRender, DebugEnv" *would* run there. The difference is that the given blades will not accept commands from other regular jobs that only request "PixarRender". The "DebugEnv(R)" item in the Provides list causes that key to become mandatory.

This subtle restriction can be used to allow wranglers to "steer" jobs to a particular set of blades that may be "reserved" for special circumstances. If a job is misbehaving on the regular farm machines, a wrangler might add "DebugEnv" (in this example) to that job's job-level service keys (e.g. in the Dashboard's type-in field). Restarted tasks from that job would then go to machines advertising the "DebugEnv" capability.

The "(R)" here prevents regular jobs from also landing on the debug machines. Even though some regular jobs will match *some* of the blade keys they do not request the key *required* by that blade.

Note that merely adding the additional key to the *job* does **not** change the usual assignment logic: the new key becomes necessary for a match as usual, so the job only they matches blades that provide that key, as usual. However, the addition of "DebugEnv(R)" to the blade *profile* means that those blades will no longer match jobs requesting only *other* keys in the Provides list -- instead those blades will *only*match jobs whose service request includes "DebugEnv" at a minimum.