# Server Profiles

## Configuration Files

The blade.config file specifies *server profiles*, each of which may apply to one or more renderfarm hosts. The idea is to have a central configuration (policy) file that controls all machines, and to allow a single profile to apply to many machines, as necessary. New servers can be brought online and will join the renderfarm using a simple "plug and play" approach using the settings for whichever profile they match.

The blade.config file is typically copied in its entirety to the site override directory and then customized there. It contains a dictionary with two top-level key entries:

- **ProfileDefaults** - defines the default values for all profiles. The base values given here are used to initialize every entry in the BladeProfiles list (below). Each profile definition can override some or all of these defaults.
- **BladeProfiles** - is an ordered list of individual profiles that may apply to one or more renderfarm hosts. The individual entries within a profile are overrides to the values specified in the ProfileDefaults dictionary.

### Profile Matching

- At startup, each tractor-blade server introspects to find a few basic traits about itself:
  - its hostname(s) and ip address(es)
  - its CPU core count
  - its GPU count and type(s)
  - its physical memory and disk sizes
  - other basic platform information
- Each blade then requests the entire blade.config file, from the engine.
- Each blade walks through the BladeProfiles list, in order.
- The *first profile that matches* the blade's basic characteristics (hostname, platform, etc) "wins" - the blade stops searching the profile list and applies that profile as its operating definition.
- When requesting work, the server's profile name is sent to the engine, allowing the assigner to determine which jobs match that host's access restrictions and which commands match the server's capabilities.

Because of the "first match wins" policy, profile definitions that are specific to just a few machines should go near the top of the BladeProfiles list, while "catch-all" general profiles should go at the end, ensuring that specialized blades are matched to the appropriate profile.

### Required Keys

There are several "required" keys that are specified in each profile object (dictionary) in the BladeProfile array. Most of the required keys are specified in the ProfileDefaults dictionary, and overrides are applied from individual profiles. Here's an example for reference:

```
{

    "ProfileDefaults":
    {   "ProfileName": "default",
        "Provides": ["pixarRender", "pixarNRM"],
        "Hosts": {"Platform": "*"},
        "Access": {
            "Crews": ["*"],
            "NimbyCrews": ["*"],
            "NimbyConnectPolicy": 1.5
        },
        "Capacity": {
            "MaxSlots": 1,    # 0 -> use number of system CPUs
            "MaxLoad":  1.5,  # CPU load avg, normalized by cpu count
            "MinRAM":   1.0,  # gigabytes
            "MinDisk": 20.0,  # gigabytes
        },
        "UDI": 1.0,
        "RecentErrorThrottle": [5, 30, 120],
        #"CmdOutputLogging": "logfile=/fileserver/tractor-logs/%u/J%j/T%t.log",
        "CmdOutputLogging": "logserver=${TRACTOR_ENGINE}:9180",
        #"VersionPin": "@env(TRACTOR_ENGINE_VERSION)",
        # "TaskBidTuning": "immediate",
        "TR_EXIT_STATUS_terminate": 0,
        "SiteModulesPath": "",
        "DirMapZone": "nfs",
        "EnvKeys": [
            {   "keys": ["default"],
                "environment": {},
                "envhandler": "default"
            },
        ]
    },


    "BladeProfiles":
    [
        {   "ProfileName": "BigLinux",
            "Hosts": {
                "Platform": "Linux-*",
                "MinNCPU": 8,      # must have at least 8 CPUs
                "MinPhysRAM": 16,  # must have at least 16 GB
            },
            "UDI": 11,
            "EnvKeys": [
                "@merge('shared.linux.envkeys')"
            ]
        },

        {   "ProfileName": "OtherLinux",
            "Hosts": {"Platform": "Linux-*"},
            "EnvKeys": [
                "@merge('shared.linux.envkeys')"
            ]
        },

        {  "ProfileName": "our_next_profile",
             ...
        }
    ]
}
```

## Key Terms

- **ProfileName** - the name of the profile entry; each profile has a site-defined, abstract title, that should describe something meaningful to the site, such as "Linux_machines" or "8core_rack" or "server_of_last_resort".
- **Hosts** - a dictionary that answers: to which hosts does this profile entry apply?
    - As each server iterates through the profile entries, they test their own host information against the Hosts dictionary of patterns to determine if the entry applies.
    - Each entry in the Hosts dictionary represents a test, and they all must pass for the profile to apply to that blade.
    - Matching is done with glob-style string compare.
    - The pattern specified by dictionary item "Name" is compared to all of the server's hostnames, aliases, and "dotted quad" ip-addresses. The "Name" entry can be either a list, or single entry. Each entry may include glob-style wildcard chars. Note! the names and addresses entered here should be the ones seen by the engine, and secondarily the ones seen by the blade host itself; hosts using VPN tunnels or other routing may not have the same data visible on the engine and the blade.
    - The pattern given by "Platform" is matched against string generated by the following Python expression on each server:

      platform.platform() + '-' + platform.architecture()[0]

- The value specified for "NCPU" is compared (exactly) against the number of CPUs reported by the host.
- Use "MinNCPU" to test for hosts that have at least the given number of CPUs.
- Use "MinPhysRAM" to test for hosts that have at least the given gigabytes of memory installed.
- Use "PhysRAM" to test for hosts that have exactly the given gigabytes of memory installed (rounded to the nearest integer).
- Use "MinNGPU" to test for hosts that have at least the given number of GPU cards. See "GPUExclusionPatterns" in blade.config for filtering.
- Use "GPU.label" to test for hosts that contain a specific type of GPU. These are wildcard matching patterns like ["*Quadro*K620*", "NVIDIA*"]
- Use "PathExists" to test for hosts that have particular directories or files present. These could be installed applications, or possibly different fileserver access mount points. The value here should be a list of path strings, ALL of the paths must be present to match.

- **Access** - server access restrictions: which **jobs** can use this server?

    - patterns defined here are matched to job characteristics by the assigner.
    - for example {"Crews": "Project-A"} would allow only jobs spooled by user who is a member of the "Project-A" crew as defined in the crews.config file on the given type of server.
    - another example {"Crews": "bob"} would allow only jobs spooled by user "bob" to run on the given type of server, e.g. bob's desktop. You can specify a username as a crew, without defining the crew in the crews.config file.
    - The crews definition can be a single value or a list of values. A list would be defined in square brackets {"Crews": ["Project-A", "Project-B"]}.
    - If not defined, the default operation is to allow all valid users access.
    - order of operations:
        a. job is spooled with certain characteristics: user, priority, metadata, etc.
        b. when the job becomes active (loaded into the assignable queue), the loader compares the job traits with the Access field on all current server definitions. It prepares a map of which types of servers can run each job - a priori.
        c. if the definition file changes, a signal to the tractor-engine causes a reload of the file and reprocessing of the access map. Similarly, servers using an old definition (detected via serial number in their request), are told to refetch the new file.
        d. when a server makes a request for a task, the server's type is used to filter out certain jobs via the access map.
    - The Access dictionary provides special access restrictions on who can change the NIMBY settings for blades using that profile. There are two basic controls: The "NimbyCrews" list defines which user crews have permission to change the setting, when change requests are routed through the engine (the default). The "NimbyConnectPolicy" setting controls whether scripts can connect directly to blades to change the nimby state, or whether they must make their requests indirectly through the engine; use a value of 1.5 or greater to allow authenticated nimby requests through the engine only.

- **Provides** - an array of **abstract service key names,** enumerating exactly which types of commands the blade can accept. For example, some particular software tool may only be installed on small subset of all farm machines, so a name representing that application might be added to the Provides list on those blades, indicating that it is installed there. Jobs that want to invoke that tool would need to have the same key name added to their RemoteCmds -service expressions, indicating that they *require* blades providing that capability. The actual key names used are arbitrary, there just needs to be an agreement on the names between the profile definitions and the jobs. For example, by convention, blades having Pixar's RenderMan renderer installed are usually indicated with the "PixarRender" key. The upper/lower case of the key spelling is ignored during comparisons.

    - If the job as a whole passes the Access tests (above), then individual ready commands are examined to see if they match the server's Provides list of service keys, for example: "Provides": ["pixarRender", "pixarNRM"]
    - The key names in the Provides list are typically just simple keywords. However, there are a few advanced annotations that can be applied in unusual circumstances.
- **Capacity** - a dictionary of constraints that cause the server to stop asking for new work.

    - "MaxSlots": (int) - maximum concurrent commands on the server. The special value '0' (zero) can be used to automatically constrain MaxSlots to the numberof CPUs found on the host (this is the default).
    - "MaxLoad": (float) - keep asking for tasks while loadavg is below this threshold.
    - "MinRAM": (float GB) - keep asking for tasks while free RAM is greater than this minimum.
    - "MinDisk": (float GB) - keep asking for tasks while free disk space is greater than this minimum. This key accepts two value formats:

      "MinDisk": 3.75,

      Or:

      "MinDisk:[3.75, "/scratch"],

      The second form names the disk to check.

- **"EnvKeys"** - a list of environment key dictionaries.

```
[
   {
      "keys": ["default"],
      "environment": {
          "REMOTEHOST": "$TR_SPOOLHOST"
     },
     "envhandler": "default"
   },
   {
      "keys": ["prman-*"],
      "environment": {
          "RMANTREE": "/opt/pixar/RenderManProServer-$TR_ENV_RMANVER",
          "PATH": "$RMANTREE/bin:$PATH"
      },
      "envhandler": "rmanhandler"
   },
   {
      "keys": ["JOB=*", "TASK=*", "SCENE=", "PROD=*"],
      "environment": {
      },
      "envhandler": "productionhandler"
   },
]
```

Each dictionary is a named *environment* and defines a list of environment *keys* which are matched against the incoming envkeys from the job to be launched.

The *environment* entry contains overrides to the default environment variables in which the command will run. The *envhandler* key contains the name of a Python environment handler that will operate on the command prior to being executed. The envhandlers can be extended by adding modules into the directory defined by the SiteModulePath entry. See the envhandler documentation for more details.

- **SiteModulesPath** - Defines a directory reachable by the blades which will contain additional python environment handlers. See the envhandler do cumentation for more details.
- **CmdOutputLogging** - destination for launched command output logs See the separate logging discussion for details on this setting.
- **VersionPin** - Specifies a particular tractor-blade "module version" that should be running on blades that are using the given profile. Those hosts are "pinned" to a particular version of the blade software. This setting is typically only given in the *ProfileDefaults*section, as a way to **push a blade update to the whole farm,** pinning all machines to a uniform blade module version.

  When an administrator reloads blade.config, then blades will inspect this setting in their profile. If a blade discovers that it is not running the correct module version, then the it will download the indicated module from the engine and *restart itself using the new module.* The blade only fetches a prepackaged version of the core tractor-blade python module itself, it does not update the entire Tractor install area, nor the bundled "rmanpy" python interpreter or other Tractor tools; a full install on the blade host is necessary to upgrade those other components.

  The named module patch package must already be present on the site tractor-engine host, typically placed there by an administrator. Each full Tractor product install contains the "factory" blade module package for that release; other blade module packages may be made available through customer support channels. These packages are self-contained zip-format modules, using the ".pyz" filename extension.

  When a blade requests a pinned module download, the engine will look for it in two places. First it looks in the engine's "--configdir" directory, that is: the directory on the engine host containing other site-modified configuration overrides. Secondly it looks in the installed product directory for the shipped package that matches the current release.
- **TaskBidTuning** - When set to "immediate" the blade will ask for new work immediately upon the (successful) exit of a prior task, or when a new task has been launched and additional slot capacity is still available. Otherwise blades will wait more conservatively between requests (minsleep secs). The "immediate" mode will cause blades to cycle through any waiting fast-running commands very quickly.
- **TR_EXIT_STATUS_terminate** - Controls whether scripts that emit "TR_EXIT_STATUS nnn" directives are left to eventually exit on their own with the given exit status code override, or should they be actively killed by tractor-blade if they don't exit promptly. Use 0 (zero) to wait, or 1 (one) to actively kill them.
- **RecentErrorThrottle** - a setting like [5, 30, 120] specifies that 5 errors on the same blade within 30 seconds will cause a 120 second hiatus in new task assignments to that blade. An "error" in this context means that a command launched by the blade exitted with a non-zero exit status code or unexpected signal due to a crash. When -1 is given as the third parameter in the list (replacing "120" in the example above), then blades that hit the error threshold will place themselves into a "nimby" state instead; in this state they will not receive new task assignments until the nimby state is unset manually by an administrator via the Dashboard or tq. Note that this setting can either be applied to all blades by putting it in the "ProfileDefaults" dictionary, or it different settings can (or overrides to the default) can placed separately in specific profiles.
- **DirMapZone** - Defines the zone used in directory mapping for cross platform support. The blades initialize with a default DirMapZone (Windows=" unc", Linux and Mac="nfs"). The value in the blade profile overrides the internal default.
- **UDI** - the Universal Desirability Index
  - This is an abstract, site-defined rating value that is used to help select "more desirable" servers when jobs are spooled and there are blades waiting for work.
  - Currently, UDI only applies when there are idle blades; if all servers are busy then tasks are assigned to whichever blade becomes available next.
- **NIMBY** - Adding NIMBY to a blade profile will define which jobs can be picked up by a blade. Settings include:

    - **1** - will only accept local jobs spooled from that blade's host.
    - **0** - will accept jobs from anyone
    - **username** - will only accept jobs from the named user.

  Note that "NimbyCrews" setting in the "Access" dictionary (described above) controls which users, if any, are allowed to change this setting dynamically through the dashboard, or other session-validated client. Note also that when providing any "Access" override in a particular profile definition, that the entire set of desired Access key-value pairs need to be specified, the unspecified keys will **not** be inherited from the Access values in ProfileDefaults.

- **GPUExclusionPatterns** -- exclude certain GPU types from matching and counting consideration. Note that this keyword is only valid in the ProfileDefaults dictionary, and GPU filtering is performed prior to any per-profile match testing. Background: A given profile can match specific hosts based on several criteria in the "Hosts" clause, these can include the count and type of GPU. Some hosts contain "uninteresting" virtual or underpowered GPUs that should always be excluded from consideration, PRIOR to the profile matching pass. Use the "GPUExclusionPatterns" list here to enumerate the makes/models of GPUs to be skipped in counts and matches. Note that "GPUExclusionPatterns" is restricted to the ProfileDefaults block only (here), it is ignored inside individual profile definitions since GPU counting occurs prior to matching. Each item in the list is a simple "glob-style" wildcard pattern, and patterns without '*' or '?' will be treated as "*TEXT*". "GPUExclusionPatterns": ["QXL", "Standard VGA "],