

Frequently Asked Questions

Table of contents

1 Questions.....	2
1.1 1. General.....	2
1.2 2. CTL: So what?.....	2

Questions

1. General

1.1. What is CTL?

CTL is a flexible distributed control dispatching framework that enables you to break management processes into reusable control modules and execute them in distributed fashion over the network. CTL helps you leverage your current scripts and tools to easily automate any kind of distributed systems management or application provisioning task. It's good for simplifying large-scale scripting efforts or as another tool in your toolbox that helps you speed through your daily mix of ad-hoc administration tasks.

1.2. Licensing and Pricing

CTL software is free and open source. See the license information [here](#). There are no hidden "gotchas". Download it. Try it. Use it.

1.3. Where do I download CTL builds?

Information for getting binaries and source code is available in the [download documentation](#).

1.4. What is the relationship between the CTL project and the AntDepo project?

Both are ControlTier projects. [AntDepo](#) and CTL share the same code roots. CTL is a re-factoring and enhancement of the original AntDepo code base. While AntDepo is primarily used as a component within the larger [ControlTier Build and Deployment System](#), CTL is designed to be fully usable as a standalone tool. In the future, CTL will replace AntDepo within the ControlTier Build and Deployment System.

1.5. Why the name CTL?

The name CTL is not an acronym. To understand our inspiration think of what [apachectl](#) does for Apache (provides a set of control operations to manage the httpd service). Well we've taken that concept to it's ultimately distributed and universally reusable conclusion. We have extended the "server control interface" idea to include the definition of any control task, be it for managing the lifecycle of a service (like httpd) or creating an interface to any arbitrary utility.

2. CTL: So what?

2.1. "My management scripts have worked all these years. Why learn and use a new tool?"

It's true; the typical state-of-the art in distributed system and application management is a set of home grown shell scripts. Management scripts almost always start off small, simple and to the point, but over time become inflexible and complicated behemoths.

It's worth learning to use a tool like CTL because it offloads some of the complexity from your script writing, and being a domain-specific framework provides many important [features you can leverage](#). As a tool, CTL provides utility out of the box. As a framework, it promotes standardization across your key automation procedures.

2.2. What's wrong with my age-old looping script approach?

It always starts the same way. You want to run several commands across a set of hosts so either typed straight into the console or written into a quick and dirty shell script, you iterate over a list of hosts, and for each one, perform one or more actions.

```
list="host1 host2 host3"
for host in $list
do
  ssh $host action-1
  ssh $host action-2
  ...
  ssh $host action-N
done
```

Of course, there's nothing inherently wrong with a looping control structure to execute actions across a set of targets. It's just that as time goes on lots of new requirements and challenges creep in and the importance of the procedure itself as a dependency to your operation increases. Here are several observations that eventually will become apparent:

- The solution started off as a temporary measure and so wasn't given a rigorous design or implementation.
- Error management is absent or very rudimentary. If an action fails does the loop script fail right away? Does it fail the set of actions for that host and continue? Is the error even noticed?
- Loop script becomes hard to scale up. As the list of hosts grows and/or the number of actions multiply or are long and slow running, how do you execute things in parallel?
- Script configuration requires editing the script itself. What happens if you need to change the list of hosts to target action? Do you edit the script and re-assign a variable? Do you have multiple variables, each defining a different set of hosts? Do these different variables represent a class of hosts, by the role they play, or their operating system,

architecture or some other logical grouping? In any case, this configuration should be separated from the loop script, externalized in some other source.

- Solution inflexible and cumbersome because after beefing up the loop in an attempt to make it more robust, most of the code has nothing to do with the task-specific activity. It's not unusual to see a 1 to 20 ratio between the lines of code that are task-specific and the bulk of code that takes care of the dispatching, coordination, and error management logic. So instead of just focussing on the task-specific problem, you're worried about scripting distributed dispatching logic. Yes, it can be fun but it's just work for the means to the end.

There are several common anti-patterns that emerge:

- The "ball of mud"
- Mixes up coordination logic with task-specific procedure
- Too many hard coded values
- Too much data and code mixed together
- Important data is buried as user variables
- Difficult to flexibly classify sets of hosts