Understanding iApps

Fred Slater, 2016-03-03

Understanding iApps

iApps are powerful tools. Used by roughly one third of all F5 customers, they perform input validation and apply complex business logic for configuring a wide variety of applications. They hide complexity, sometimes driving hundreds of configuration parameters with just a handful of input values. They also provide deployment guidance, reducing the need for documentation and training. But iApps are often misunderstood and misused, sometimes with frustrating consequences.

iApp, not Wizard

It is helpful to understand the difference between an iApp and a wizard. A wizard is usually a script with a GUI. It is a tool that accepts a set of user inputs and performs a one-time procedure. When run twice, a wizard performs exactly the same steps during its second run as it did during its first. Although an iApp is also a script with a GUI, it behaves differently on re-entry. Unlike a wizard, an iApp maintains a relationship to the configuration that it generates.

iApps have 5 critical properties:

- iApps always act atomically. The result of deployment is always either the entire intended configuration or none at all.
- iApp-driven configuration objects are marked so the iApp can track them throughout its lifecycle. A visualization of these configuration objects is presented in the popular TMUI “Component View.”
- iApps protect the configuration from accidental changes. iApp-driven elements may not be changed via the UI or CLI except through the iApp.
- iApps support re-entry. Since the iApp framework tracks the configuration objects that it manages, it can be intelligent about which elements are touched during reconfiguration.
- iApps automatically perform cleanup on deletion. The housekeeping is automatic, and the iApp author does not need to (and is not allowed to) write any delete-time code.

Mark and Sweep

On initial deployment, an iApp builds a configuration as directed by the TMSH commands in the iApp’s implementation section. On reconfiguration, the iApp framework again runs the implementation, but does not immediately submit the result to BIG-IP. Instead, it first compares the desired (new) configuration with the existing (old) configuration. All similarities are dropped from the workload. If existing elements are missing from the desired configuration, the framework automatically prepares delete requests for those elements. In the end, the changes submitted to BIG-IP are the minimum necessary to accomplish the desired configuration. TMSH “create” commands used to build config on the initial deployment are automatically changed to “modify” on re-entry. This mechanism, known as “mark-and-sweep,” reduces a scripted change set to its net effect, often allowing iApps to reconfigure BIG-IP with no disruption to data plane traffic.

Strict Updates

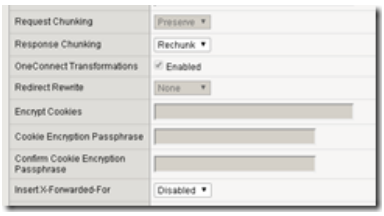
As noted above, an iApp tags all of the configuration objects under its control and BIG-IP prevents users from modifying these objects outside the iApp. This restriction, called “strict-updates”, can be disabled. The negative consequences of this action are often underestimated.

To illustrate the typical frustration, consider a simple iApp that accepts 2 inputs and constructs a virtual server with a custom HTTP profile:

```
presentation {
    section virtual {
        string ip required
        choice xff { “enabled”, “disabled” }
    }
}
```

```
implementation {
    tmsh::create ltm profile http ${tmsh::app_name}_http \
        insert-xforwarded-for $::virtual__xff
    tmsh::create ltm virtual ${tmsh::app_name}_vs \
        destination $::virtual__ip:80 \
        profiles add \{  ${tmsh::app_name}_http \}
}
```

Suppose you deploy this iApp with X-Forwarded-For enabled, then disable strict-updates and edit the HTTP profile directly as shown:



Now, re-enter and re-deploy the iApp without changing any inputs. What is the result? You might expect your manual customization of response chunking to remain in place, because there is no mention of response chunking in the iApp code. In this case, however, iApp re-deployment causes chunking to be set back to its default value.

Consider another change, this time to the virtual server:

Disable address translation, then re-enter and re-deploy the iApp again. What is the result this time? You might expect your customization to be overwritten as it was in the previous example. In this case, however, address translation remains disabled. The TMSH “modify” command does not handle defaults consistently across all key-value pairs. This makes tampering with an iApp-enforced configuration especially hazardous.

Right Use

Hopefully, this article provides some insights that will help you use iApps to your best advantage. An iApp is a scripted management tool that is dedicated to an application for life. If you find yourself wishing to disengage the iApp and tamper with its configuration, instead consider modifying the iApp code to accomplish your goals. If you absolutely must disable strict-updates, do so with crystal-clear expectations for managing that application going forward.

F5 Networks, Inc. | 401 Elliot Avenue West, Seattle, WA 98119 | 888-882-4447 | f5.com

F5 Networks, Inc.
Corporate Headquarters
info@f5.com

F5 Networks
Asia-Pacific
apacinfo@f5.com

F5 Networks Ltd.
Europe/Middle-East/Africa
emeainfo@f5.com

F5 Networks
Japan K.K.
f5j-info@f5.com