realtimepublishers.com™

The Definitive Guide™ To

Exchange Disaster Recovery & Availability

Paul Robichaux

Jim McBee, technical editor

XOsoft™ JUST KEEP WORKING

Copyright Statement

Chapter 4: Availability Building Blocks—Validation, Testing, and Deployment

The preceding chapters have all focused on technologies that you can use to improve the recoverability, availability, and resilience of your Exchange infrastructure. However, technology alone won't give you a lasting improvement. This chapter describes how you can validate, test, and deploy your disaster recovery, high-availability, and business continuity solutions so that you'll reap more than just temporary benefits.

Testing, Testing, 1, 2, 3…

An untested solution is no solution. This principle is an effective guide to use throughout your Exchange deployment effort. If you're trying to improve your messaging availability and recoverability and you haven't demonstrated that your solution actually makes things better, you haven't solved anything—you've simply tossed money around to no good effect.

In this case, "testing" isn't really the correct term. Every administrator has an intuitive understanding of what it means to test something, but that understanding differs quite a bit between individuals. What you really need for a solid highly available/disaster recovery design and deployment is more than just the usual "smoke test" of trying a procedure or technology to ensure that it doesn't fail.

There are three components to the kind of testing you need to perform, one of which includes the common perception of what testing entails:

- Validation—Validation requires you to ask, and answer, two probing questions: Do the requirements for my Exchange design reflect my business requirements? In other words, have you accurately specified what the completed system should be able to do? The second question is perhaps easier to answer: Can I demonstrate that my design meets the requirements? Simply put, the validation phase is all about proving that your design is correct, both in design and in implementation. This phase encompasses aspects such as performance benchmarking of your servers and their storage subsystems.
- Testing—There are actually two kinds of tests of interest to Exchange administrators. Unit testing refers to testing individual components, and integration testing refers to testing multiple components and the intercommunication between them to verify that they work as designed. From an Exchange viewpoint, you might think of unit testing as checking an individual server to ensure that users can log on to it, and integration testing as verifying that the server interoperates with the other servers in its routing group.
- Deployment—This component involves putting the planned system into production. A simplistic deployment plan might call for plugging in items to determine whether they work; a more realistic plan includes functional tests and performance checks as part of the deployment to ensure that the implemented system still meets the validated requirements.

This description makes testing sound more complex than you might expect; testing can certainly be a very complex and time-consuming endeavor for large-enterprise Exchange deployments. However, the cost of *not* doing a thorough validation, testing, and deployment pass is greater than the time and effort this step requires. Thorough testing is critical to ensuring that your Exchange systems deliver the level of security, robustness, and performance that your business requires.

This chapter assumes that you're testing and validating a design that you got from somewhere—perhaps it was developed in-house or it was developed by an outside consultant. The source doesn't matter, and the chapter won't dwell on specific design features. Instead, the focus will be on explaining how you can apply the validation, testing, and deployment cycle to the process of improving your Exchange systems' availability.

Validation

The accuracy of results of any kind of endeavor play a critical part in making that endeavor worthwhile. So it is with validation. The software engineering community has a simple rule of thumb—making a change in the design is about ten times more expensive than making a change to the requirements, and making a change to the actual software is ten times more expensive than changing the design. The cost differential for Exchange probably isn't as steep (after all, two orders of magnitude is quite a penalty), but it definitely exists.

The Value of Verification and Validation

In true mission-critical environments, such as NASA, one of the cardinal rules for software development is that you don't rely on your own, or your own company's, testing procedures. There is always an independent verification and validation (IV&V) team whose job it is to *validate* that the requirements for the software are correct and *verify* that the software actually meets those requirements. The reason for having a separate team to do so is obvious. Although it provides an opportunity for finger-pointing between the development and IV&V teams, it also means that the developers can't fudge things, intentionally or not. The IV&V group acts as a watchdog to ensure that the right software is being built, and that it will work as designed. Although their presence is invariably accompanied with lots of groans from the architects and developers, in the end, the IV&V team contributes a great deal to a successful delivery of software.

For Exchange design, you can apply this same principle. If you hire a consultant (or company) to design an environment for you, hire a *different* consultant to review the design before implementation. Doing so provides a sanity check from a dispassionate third party. Even if you're responsible for the design yourself, having an outside party check over it can be a valuable way to find faults before deployment. This review should include validation, testing, and review of your deployment plan. In particular, whoever does your IV&V pass should take great pains to ferret out any hidden assumptions or implicit service level agreements (SLAs). Every place a number is cited—whether it's a cost, a performance metric, or a threshold—there should be justification for that number. Without that, it's difficult to validate the accuracy of the plan.

If it's not possible for you to hire an outside consultant, you should still be able to find someone to help you evaluate your design. Local user groups are a good place to look for peers who can cast an eye over your proposed configuration; you can also ask for help in any of the many excellent Exchange support resources out on the Internet. Even a cursory review by someone who has no stake in the outcome can turn up issues that you'll want to address before your actual deployment begins. If nothing else, you can probably pick up some valuable tips from NASA's official IV&V site at http://www.ivv.nasa.gov.

Are the Business Requirements Adequately Specified?

Two phases to validation were mentioned earlier. Let's start with the first one, because it's often the most difficult for administrators to manage. The reason for this difficulty is that the validation phase depends on being able to answer questions that may not be within the normal scope of your job.

Simply put, your goal in this phase is to figure out whether the proposed system will meet your business needs. Let's take a simple example. Say you have 2500 mailboxes in your Exchange organization, averaging 500MB in size each. You expect mailbox sizes to grow at 15 percent per year for the next 3 years, and you expect the number of mailboxes to grow at a maximum of 8 percent in each of the next 3 years. Your SAN vendor proposes a SAN with 2TB of total storage. This amount won't work because the mailbox size growth alone will result in about 1.9TB of data (2500 mailboxes @ 0.5GB each = 1.25 TB * 1.15 * 3)—a 2TB SAN doesn't leave you any room to add mailboxes, perform maintenance, or grow even a tiny bit above your estimates.

Of course, you can argue about the specific numbers used in this example. Perhaps a 15 percent growth rate is too high, or maybe 8 percent is twice the year-over-year mailbox growth you've had in each of the past 3 years. Regardless, you should always build in room for error and additional growth. Remember that the goal is to come up with a credible estimate that preserves adequate headroom for the future. You might observe that the cost per gigabyte of disk storage keeps dropping and argue that it makes more sense to defer buying storage until you actually need it. The problem with that approach is that storage *management and integration* is what costs the real money nowadays; the initial acquisition cost of the storage is low by comparison. As a further argument, remember that the storage you buy at initial deployment time can be tested, integrated, and provisioned without any interruptions in service; storage that you add later may not be quite so easy to get going.

You can do similar back-of-the-envelope calculations (applying an appropriate adjustment factor at each step) for every other aspect of Exchange design, including the number of mailboxes you put on a server, the number of servers you maintain, and how much planned and unplanned downtime your organization can tolerate.

This last factor is worthy of a bit more explanation—as a messaging administrator, you probably don't have data about how much a messaging outage actually costs in business terms, but you probably know how much downtime is tolerated just based on your own anecdotal experience in the organization. It's best to work with someone who *does* know so that you can accurately estimate how much you can, or should, spend to get to a certain availability level. There are many costs associated with downtime, both tangible and intangible; the exact costs associated with an outage will vary according to your line of business. For example, lost sales would be a major concern to a company that takes client orders via email, but not for a university. For most organizations, the largest cost associated with downtime is lost productivity, and to quantify that you'll need help from other parts of the organization.

However, even if you can't do this, knowing how much downtime is allowed will help you make appropriate decisions. If you've followed the advice from Chapter 1 to consider what your SLAs are and should be, you'll have a pretty good idea of what these figures look like, and you'll be able to provide information such as "If we spend $*x*, we should have no more than *y* hours per year of downtime." At that point, deciding whether it's a good idea to spend that amount of money is no longer your problem; instead, the people who sign the purchase orders will decide what to do.

There isn't a standard chart that illustrates the amount of money spent to obtain a given increase in uptime because the data needed to generate such a chart would vary widely between organizations. *Your* uptime and *your* budget will dictate what such a chart would look like for your organization. It's safe to say, however, that getting the first couple of nines (for example, moving from a 95 percent planned uptime SLA to 99.0percent or 99.9 percent) will cost less than adding additional nines.

Of course, the natural tendency in this phase is to specify absolutely as much hardware as you can afford in the hope that doing so will give you enough performance and capability to meet your needs. This route is safe in many cases, but it might require you to forgo other things that you would like to be able to implement. For example, spending big bucks on a clustered server for 200 mailboxes means that you're spending money on a cluster node and shared storage system that could be spent on additional redundancy and resiliency elsewhere. The opposite is also true; you may be tempted to design to the absolute minimum standard that you see yourself needing in an attempt to keep some budget money free for other purposes. The problem with doing so is that in a year or two, when you need to expand, you may have trouble finding compatible hardware. If you buy two identical servers now, you can cluster them later. If you buy one server now and another next year, they'll have enough dissimilar components to make clustering them impractical (and, almost certainly, unsupported by Microsoft).

Does This Design Meet the Stated Requirements?

Once you decide what your requirements *are*, you're ready to delve into the second validation phase: finding out whether the proposed design meets those requirements. There are several dimensions on which you can evaluate a design, all of which are important. However, they're not *equally* important. Which dimensions take precedence in your environment will depend on the environment; for example, equipment cost is critical for low-budget organizations such as schools, while SLA compliance might be most critical for a large corporate deployment. A corporation undergoing rapid growth may value scalability above all; a company with a small IT staff may value a simple turnkey solution, even if it costs a bit more. Organizations that rely heavily on email for near real-time information with customers are going to focus on an excellent Recovery Time Objective (RTO). The important point to remember is that most organizations fall somewhere in between these extremes. Virtually no one has an unlimited budget, so cost is always an issue; at the same time, many organizations find that they have a strong business justification for maximizing one of the other dimensions.

The major dimensions you should analyze when validating the suitability of your design include:

- *Acquisition cost*—This cost seems straightforward enough, but it can actually be quite tricky. You must consider the total cost of whatever you're buying—the walk away cost. In other words, when you walk away, how much lighter is your wallet? When you're examining a design for its cost basis, be sure to factor in any software or hardware add-ons that you need. For example, if you've moving to a clustered Exchange configuration, you'll probably need to buy an upgrade to make your antivirus software cluster aware. It's easy to focus on the cost of big-ticket hardware items and miss the smaller items that can actually push your design over its budget limits.
- *Maintenance/upkeep cost*—The walk away cost is just the beginning! One important aspect of validating your design against its requirements is figuring out how much it will cost to maintain once deployed. Failing to factor in this cost up front can lead to a great deal of hate and discontent when the time comes to renew annual maintenance agreements, buy more disks, or take other actions that will require more money than you spent—or planned for—up front. Of course, as you consider the maintenance cost of your proposed solution, don't forget to include the people cost. If you're moving to a technology (such as SANs or clustering) that's new to your organization, it's going to cost you money to acquire organizational competence, either through training or hiring. Of course, the more expensive skills your administrators amass, the more difficult (and more costly) it will be to keep them around, and that's another cost to be considered.
- *Standards conformance*—Most large organizations have hardware standards that govern what you can buy. Sometimes these standards are optional; other times they are vigorously enforced in an effort to keep good control over the number of different configurations. From an Exchange standpoint, the fewer different configurations you have to support, the better. Most organizations that are deploying or upgrading Exchange prefer to buy a quantity of identical servers up front instead of acquiring them piecemeal. Thus, you may spend more money up front (thereby stressing the acquisition-cost dimension) but save money later by keeping your environment and management tools consistent. Even if your organization doesn't have fixed standards, you should review Microsoft's Windows Hardware Quality Lab (WHQL) hardware compatibility list to ensure that your equipment appears on it. This is particularly important for clusters and SANs. Don't buy a cluster or SAN configuration unless it's certified by WHQL.
- *Scalability or growth capability*—Even if your requirements don't explicitly include growth targets, you must consider the scalability of your proposed solution. For example, if you're thinking of planning a cluster solution, how many concurrent users can each node handle? Will the nodes be able to accommodate however much growth you expect to have between the time you buy them and the time they're retired or decommissioned? If you expect to go through periods of high growth (perhaps your company is expanding into China or merging with another larger company), does your design account for it? Shortfalls in performance may require you to spend more money later, which affects how suitable the design is to your cost requirements or limits.

- *Performance*—Perhaps unsurprisingly, performance is where the majority of administrators get into trouble. The trouble that arises here comes from two primary sources. First is failure to account for growth—the design that's adequate for today's mail load may not be able to handle the future. The other, sad to say, is an unfortunate comment on the state of the industry—you should never accept any vendor's performance claims at face value. It doesn't matter which vendor, or what their claims are. Reputable vendors generally do a good job of generating honest benchmarks for their gear, but you can't assume that their assumptions and constraints match your environment. The best way to ensure that your performance requirements will be met is to give the vendor an opportunity to let you run your own set of benchmarks on representative hardware. If they won't do that, consider carefully whether you should give them your money.
- *RTO*—An acceptable high-availability or disaster recovery design will naturally include requirements that specify the RTO for the deployed system. In fact, one of the key drivers behind your deployment may be a requirement for a shorter RTO. The limiting factor in most recovery scenarios is how fast the recovered data can be retrieved from its storage medium. For example, say your RTO is 4 hours, and you're using tape as your primary recovery mechanism. Your requirement is thus to be able to implement a complete recovery (including all the factors discussed in preceding chapters) within that time—*not* just to be able to pull the data off tape. Accordingly, you'll need to verify that your servers are sized such that you can back up and restore their data within your RTO, and that your backup and restore mechanism can handle the associated volume of data in a timely manner.
- *RPO*—Deciding whether a specified configuration will allow you to meet your RPO requires you to understand your RPO (easy) and how the proposed solution will meet it (not so easy). If you're using replication, clustering, or SAN technology as the basis of your RPO solution, plan to spend time in the test lab to make sure that you can actually recover to the specified point in time using the selected components.

The synthesis of all these factors is whether your solution will meet your SLAs. Determining whether such is the case is a difficult task because there are competing factors that you must counterbalance. For example, you can spend more money to get a shorter RTO—but does that match your business and cost requirements? Are your SLAs realistic? If not, adjusting them before deploying your proposed solution can save you a lot of money and trouble—and maybe even your job.

The Way Microsoft Recommends Calculating Server Size

In the *Exchange Server 2003 Performance and Scalability Guide* (http://www.microsoft.com/technet/prodtechnol/exchange/2003/library/perfscalguide.mspx), Microsoft recommends using two metrics for calculating the capacity of your servers: *megacycles/mailbox* and *IOPS/mailbox*. These may seem like odd metrics, but they can be quite useful. Microsoft defines them in the guide as:

- *Megacycles/mailbox*—This metric measures the raw CPU usage required per mailbox during peak periods. For example, if a user uses one megacycle/second during peak operation and there are 1500 users on the server (which equals 1500 megacycles/second), a single 2GHz (2000MHz) processor will operate at 75 percent CPU utilization.
- *IOPS/mailbox*—This metric covers the number of I/O operations per second (IOPS) required for database (not transaction log) disk usage, per user, during a peak period. For example, if each mailbox uses 0.5 database IOPS during peak activity and there are 1000 users homed on the server, there are 500 database IOPS. IOPS/mailbox metrics are based on random read/write Exchange database I/O operations.

To calculate either of these metrics, you need to know how many users or mailboxes you have on a server. There's some dispute over the preferred way to make this determination. One school of thought is that you should use the number of mailboxes on the server; the other is that it's better to use the number of active users over the measurement period (which is why the active user count performance counter is valuable). It doesn't really matter which approach you take *as long as it's consistent*. If you use mailbox counts to figure out the sizing profile for one server, you must use the same method on other servers too.

Once you have that information, the actual metrics are easy to calculate. Megacycles/mailbox can be calculated based on the CPU usage, the number and speed of processors, and the number of mailboxes: multiply the first three and divide by the mailbox count. Let's say that you have a 60 percent CPU load on a dual-2GHz machine that hosts 2000 mailboxes. That gives you (0.6 * 2 * 2000) / 2000 = 1.2 megacycles/mailbox.

IOPS/mailbox is easier; simply divide the average disk transfers per second by the mailbox count. Of course, that requires you to measure the average disk transfers per second, and to know how many mailboxes you have on each server.

Armed with these two metrics, you can then use them to calculate the maximum number of users that you can put onto a particular server configuration, following the procedure outlined in Appendix C of the *Exchange Server 2003 Performance and Scalability Guide*. You should initially perform this calculation on the servers you already have so that you can compare your existing infrastructure with what you're planning for the future. This requires some extrapolation; for example, if you're going to have a quad-Xeon 3GHz platform as the basis for your new mailbox servers, and you want to keep it to below 70 percent load, that lets you figure out how many mailboxes you can tolerate with the same megacycles/mailbox as the example above: (0.7 * 4 * 3000) / X = 1.2, so X = 7000. Of course, that's probably too many mailboxes for the proposed storage system, so that might be a guideline that you're spending too much money on CPU horsepower for that proposed configuration.

Testing

Paper validation is straightforward, but the ultimate test is to *try* the proposed configuration. This testing isn't always possible, particularly in large enterprises in which there isn't a realistic way to replicate what the production environment looks like. When it is possible, though, there is really no substitute. Obviously, you can't completely duplicate your production setup for test purposes, but there are several things you can do to help make your testing more realistic.

Although software developers often make a distinction between unit and integration testing, that distinction is harder to draw for Exchange. Most of the test tools available to you actually are integration testing tools, and you normally use them to test the overall performance of your Exchange servers—performance that in turn depends on DNS and Active Directory(AD).

Testing Tools

The first step toward building a solid test plan is understanding what testing tools are available. Microsoft makes three primary tools that you can use for load testing and simulation.

LoadSim

First is the venerable LoadSim (or, more precisely, the Microsoft Exchange Load Simulator). LoadSim is a benchmarking tool; it just so happens that you can use it to simulate load on your servers, but with some important caveats. You configure LoadSim with a given number of MAPI clients, using a set of profiles that you can adjust to reflect the actual amount of load generated by your users. There are several profiles included with LoadSim, and you can adjust the parameters used in a given profile. LoadSim will then generate synthetic MAPI transactions to imitate clients opening, sending, reading, and deleting messages. You can configure two testing periods: daytime (when users are presumably most active) and nighttime, and you can adjust the length of these periods.

LoadSim is a very useful tool for determining how a given configuration reacts under load, but *that load doesn't precisely match what real users do*. If you build a LoadSim profile for 1500 users, what you're going to get is a big load of MAPI transactions that don't necessarily happen at the same time, or the same frequency, as real workloads would. For example, let's say you stick with an 8-hour daytime period. LoadSim will spread transactions out over this entire period; in the real world, there tends to be heavy load on servers when people first get back to work, with peaks and valleys throughout the day (for example, a valley during lunch, then a peak when people come back). In some environments, you might see as much as 50 to 70 percent of the total workload on an Exchange server occurring during one or two peak periods, but LoadSim doesn't address this reality; it will spread the load out across the entire time period. You can work around this limitation by decreasing the length of the "daytime" test period so that LoadSim must pack the transactions into a shorter period. In order to do that accurately, though, you need to understand where the real peaks in your mail usage are, and for that you need the baselining process described later.

In addition, because it doesn't model peaks in activity, LoadSim alone won't necessarily give you a complete view of how the storage system on your servers will perform. To work around this problem, you can (and should) use Jetstress, as described later.

However, LoadSim provides a valuable way to perform capacity validation of a hardware configuration. You start by gathering baseline data about your existing system by running LoadSim against it and tweaking the profiles and usage hours so that LoadSim's results match your observed performance numbers as closely as possible. Then you take the same LoadSim configuration and run it on the proposed new configuration to determine the performance differential.

Exchange Stress and Performance Tool

The Exchange Stress and Performance Tool (ESP) doesn't receive a lot of press because it's primarily designed to load-test Exchange's Internet protocols. In that role, you might consider it primarily as a unit-testing tool, because you can test individual protocol implementations more or less in isolation. ESP includes support for a wide range of protocols; the full list is pretty impressive:

- WebDAV (used by Outlook Web Access and Entourage for Mac OS X)
- Internet Message Access Protocol version 4rev1 (IMAP4)
- Lightweight Directory Access Protocol (LDAP)
- OLEDB, the protocol used internally by the Exchange store
- Network News Transfer Protocol (NNTP)
- Post Office Protocol version 3 (POP3)
- Simple Mail Transfer Protocol (SMTP)
- Exchange ActiveSync for mobile devices that synchronize data with the Exchange server
- Outlook Mobile Access for mobile devices that browse data but don't synchronize it

Unless you have many IMAP or POP users, this tool might not seem that useful; however, it's *very* useful for testing the performance of high-volume SMTP gateways. If you're going to have more than a few hundred messages per hour passing through your SMTP machines, it's probably worth spending a short amount of time with ESP to learn how many messages per hour your proposed configuration will support. The following example shows a simple script from the ESP documentation:

```
CONNECT
EHLO
MAIL FROM: <me>
LOOP RANDNUMBER(100, 1000)
RCPT TO: rcptRANDNUMBER(1, 20)@RANDLIST(domains.txt)
ENDLOOP
DATA RANDLIST(bodies.txt)
QUIT
```

This script sends a single message with a randomly chosen body to a random set of recipients. By wrapping this script in a larger loop and adjusting the message bodies, you can send messages in a size and time pattern that matches your actual SMTP traffic. The ESP documentation completely describes the script commands available for each protocol, so you can create arbitrarily complex scripts that match the load you actually expect users to create on your servers.

Jetstress

Jetstress allows you to wring out your storage subsystem to measure its Exchange performance capabilities. It's important to understand the distinction between LoadSim, which models user operations, and Jetstress, which uses the Extensible Storage Engine (ESE) component of the Exchange store to make lots of I/O requests that fairly accurately replicate Exchange's access patterns. Jetstress is a critical testing tool for SANs and replication configurations because it allows you to stress the storage subsystem and see how it behaves under load.

In particular, Jetstress allows you to acquire accurate performance data that reflects how many I/O operations per second (IOPS) your proposed storage configuration will give you. It doesn't do so directly; instead, Jetstress will create a database and hammer it with the number of IOPS you specify. Your goal in running these tests is to keep user performance (as measured by read and write latency) to an acceptable level. In general, Microsoft recommends that your read and write operations not have more than 20ms of latency on average, with 40ms as the do-not-exceed peak value.

In an ideal world, every SAN would dedicate physical disks to Exchange, and the SAN controller cache would always be big enough to satisfy a large percentage of I/O requests. In the real world, these things don't always happen. Jetstress is an effective tool for revealing the true performance capacity of a given logical disk configuration on a SAN; if you let it run for long enough, you'll be able to see the change in performance that occurs when Exchange detunes the SAN cache. The number of Exchange databases plays a role here too, because the number of databases and storage groups you have (and where they're located) makes a big difference in the actual disk I/O patterns.

Jetstress actually operates in two modes. A performance test checks the performance of your storage system to see how it behaves; a stress test runs for 24 hours to see whether the system will work properly under high-sustained loads. You should plan on running both types of tests as part of your test procedures.

Microsoft's documentation for Jetstress explains some factors to keep in mind when planning your stress testing:

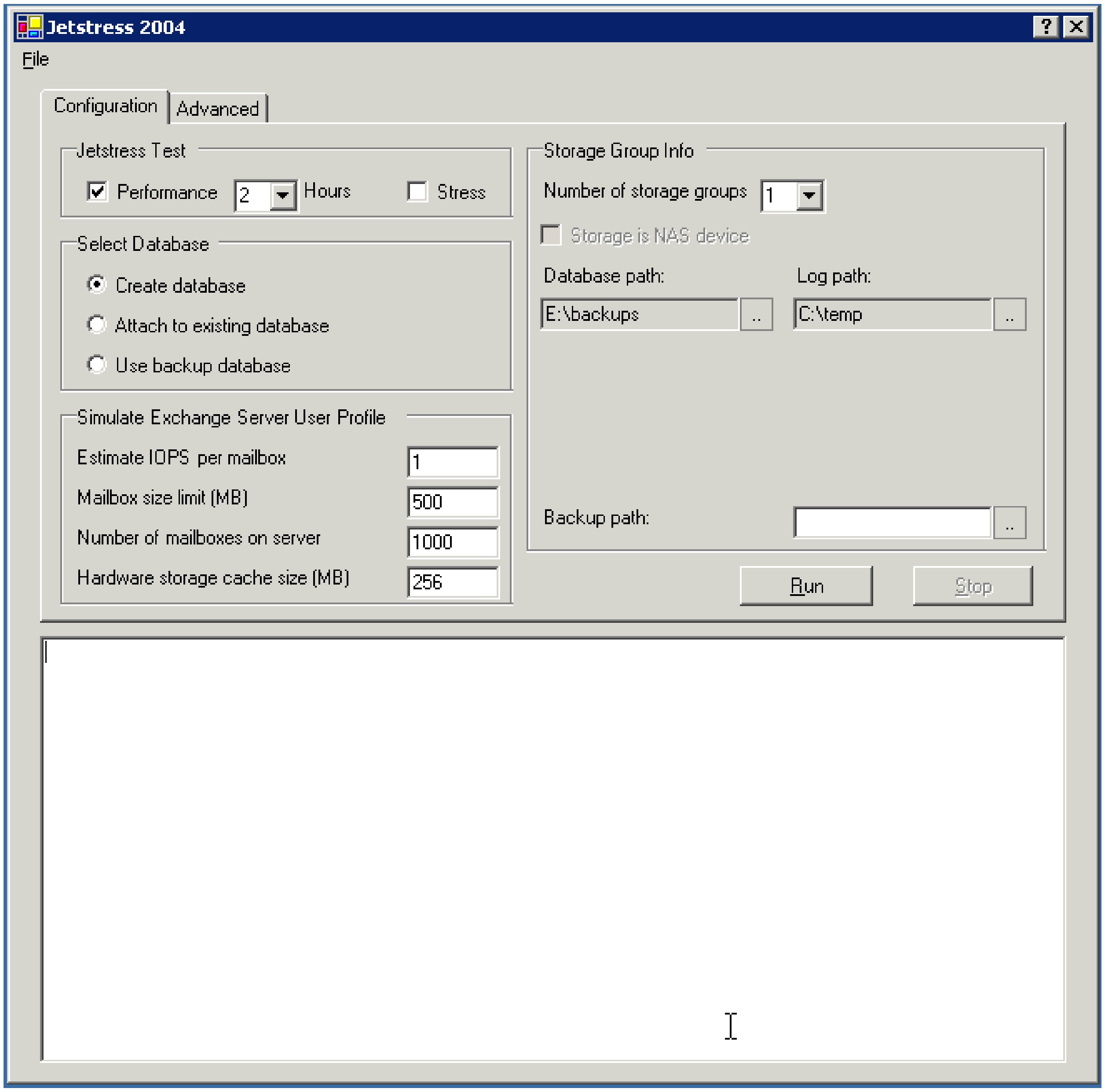
- If you're going to put multiple storage groups or databases on a disk, you should run as many instances of Jetstress as there are storage groups or databases. For instance, if you're putting each set of log files and EDB/STM files on a separate disk, and you're using two storage groups, you should have two Jetstress sessions running—one for each disk set.
- If you're testing a SAN that will have multiple servers attached to it, make sure to run Jetstress on all those servers at the same time to accurately simulate what the production load will look like.
- Jetstress expects you to specify the expected size of your databases. It then allocates twice as much space as you specify (to allow for overhead), and creates databases that are 75 percent of the projected size. This sizing gives a more accurate picture of the IOPS capability of the disks; smaller databases on the same disks would give you a distorted picture.

Another Jetstress consideration is that this tool doesn't measure anything other than disk performance. You can use it to simulate the disk load you think your users will generate, and compare that with the disk load that your storage configuration can sustain. Doing so on a mockup of your proposed system will tell you how that system will perform when you deploy it; the more interesting data will come from seeing how that performance changes as you add users and disks. When possible, it's a good idea to use Jetstress as a burn-in tool—run it for multiple consecutive 24-hour cycles to shake out any subtle performance problems or unwanted interactions between different firmware versions.

Microsoft field engineers often recommend using the command-line version of Jetstress to create databases that occupy 80 percent or so of the projected disk space. The GUI version uses 75 percent of capacity. Why? With nearly empty disks, the controller doesn't need to work nearly as hard. This is particularly true of SAN configurations that use a smaller number of larger disks to provide the desired amount of storage; as those disks fill up, the performance you see can be very different.

☞ If you're going to share a SAN between Exchange and other applications, it's a good idea to run Jetstress at the same time as your other applications are undergoing SAN tests too—doing so will tell you what will happen when a shared application makes a large set of I/O requests (as when SQL Server does a database integrity check).

When you configure Jetstress, you'll need to specify several parameters, most of which are self-explanatory (such as the length of the test period). However, there are a few parameters that are worth a more detailed mention, as Figure 4.1 shows.



Jetstress Test
[✓] Performance 2 Hours
[ ] Stress

Select Database
(●) Create database
( ) Attach to existing database
( ) Use backup database

Simulate Exchange Server User Profile
Estimate IOPS per mailbox: 1
Mailbox size limit (MB): 500
Number of mailboxes on server: 1000
Hardware storage cache size (MB): 256

Storage Group Info
Number of storage groups: 1
[ ] Storage is NAS device
Database path: E:\backups
Log path: C:\temp
Backup path:

Figure 4.1: The Jetstress configuration user interface.

- Estimate the IOPS/mailbox for the test. You can always cheat and use 1 IOPS/mailbox, which is what Microsoft calls the "heavy information worker" profile. More typically, most users consume between 0.4 and 0.8 IOPS on a sustained basis (but see the next section for information about peak IOPS consumption, which is probably a better overall way to run Jetstress).
- Specify the number of mailboxes that you want to put on the server and the mailbox size limit.
- Specify the hardware cache size if you're using a SAN or caching controller. Jetstress factors in this information when building its I/O patterns. Be sure to get this right; if you enter the wrong value, Jetstress will bias its test numbers to the wrong cache size.

To run Jetstress, you'll need to stop the IIS Admin service and all Exchange services. Doing so makes sense—having those services running means that they would be contending for disk resources against Jetstress. However, it also means that you can't run Jetstress on a production server without stopping the information store. Jetstress tests are pass/fail. During the test, the Jetstress tool monitors a range of performance counters; if the value of any of these counters falls outside the specified range, the test fails.

Testing and Data Replication

In the Jetstress documentation, Microsoft makes a special effort to describe the use of Jetstress to test replication solutions. It is important that you understand the highlights—if you're considering using a replication solution in your production system, be sure to read the documents in full.

Basically, to validate that your replication solution will work in production, Microsoft recommends that you perform two separate Jetstress passes. The first pass is to validate the performance of your storage subsystem without replication. This process is identical to what you would do if you weren't planning to use replication at all. The second pass, though, should be run with replication enabled in the configuration you're planning to use in production. This step is particularly necessary for synchronous replication solutions because the latency of the source system will increase proportionally to the latency imposed by writes to the remote system. If you're using a synchronous replication solution, you should also add a LoadSim test pass to ensure that client performance with replication enabled meets your requirements.

If you're using an asynchronous replication solution, latency shouldn't be an issue for you. However, that doesn't mean you're out of the woods. You still need to worry about the amount of bandwidth between the source and target systems because insufficient bandwidth means that updates may not be applied to the remote side in a timely fashion. In addition, during the time it takes data to be replicated from source to target, a system failure may mean that you lose any data that hasn't yet been replicated. The testing phase is an ideal time to validate your assumptions, and goals, for uninterrupted operations.

Performance Baselining

It's important to gather an adequate set of baseline performance data on your existing systems before you begin performance assessments on the proposed system. One reason is that baselining helps ensure that you're making apples-to-apples comparisons; another important benefit is that you can use the baseline data to ensure that your LoadSim and Jetstress tests are producing loads that accurately reflect the load on the current production system. For example, if your average CPU utilization on a busy mailbox server during the day is 55 percent, you would use LoadSim and tweak the client profiles so that your test run generates a similar CPU load on the existing system, then run the same LoadSim test set on the new hardware to see what its CPU load looks like.

Windows and Exchange expose hundreds of performance parameters; you can't monitor them all at once, and doing so would drown you in data if you tried. Instead, by watching a few carefully chosen counters in Performance Monitor, you can get a pretty good view of the load on your Exchange server. The following list highlights the basic set of counters to watch:

- *Average disk queue length*—This counter tells you how many disk operations, on average, are waiting for their turn. The average length should be smaller than the number of physical disks that you have. If you have three disks in a RAID-5 array, for instance, your average queue length should be less than three. However, if you're using SANs you'll need to check your SAN vendor's Web site to ensure that you're monitoring the correct set of logical and physical counters to match their recommendations.
- *Log Threads Waiting*—This counter tells you how many transaction log threads are waiting their turn to write data to disk; you can use this counter along with *Average disk queue length* to see how many pending write requests are queued at a given time.
- *Disk transfers/sec*—On both the physical and logical disk objects, these counters tell you how many total IOPS are happening on your disks. It's most important to monitor these on your Exchange database disks, although it's a good idea to also monitor your SMTP queue and transaction log volumes to get an idea of the kind of volume they're experiencing.
- *Active user count*—On the MSExchangeIS object, this counter tells you how many users are logged on to the store of a given server at a given point in time. You can use this figure in conjunction with the *Disk transfers/sec* counters to calculate the number of IOPS per active user. Note that this counter measures active users in the past 10 minutes and includes all users who have logged on to the store, including non-mailbox requests such as public folder and OAB data.
- *Memory/Available Mbytes*—This counter measures the free RAM on the server. Microsoft's recommendation is that Exchange servers always have at least 50MB of free RAM when running at normal load; a more general recommendation is that you should always have at least 10 percent of a server's RAM available.
- *Memory/Pages/sec*—This counter tells you how many virtual memory page swaps occur per second. Too many page swaps indicates that you don't have enough physical RAM; the average value should be less than 100 pages/second, and the peak value should never exceed 1000 pages/second.
- *Memory/Free System Page Table Entries*—The system page table tracks where pages are on disk. If the system runs low on available page table entries (PTEs), performance will suffer; beyond a certain point, so will stability. The minimum number of free PTEs should never drop below 5000.
- The MSExchangeIS object's *Read Bytes Clients/sec* and *Write Bytes RPC Client/sec* counters tell you how much network activity is being generated by MAPI clients; monitor them to verify that your test configuration is generating as much network activity as live operations do.
- *RPC Average Latency*—This counter should remain below 50ms; if it rises above that value, it is a good sign that something is making your RPC operations take too long.

- *% Processor Time*—This counter tells you how busy the system CPUs are with user-mode requests. You can either monitor individual CPUs or the total average CPU usage; the average is probably more useful for most baselining projects.

You should monitor data from these counters at a predetermined interval. Jetstress samples performance data at 15-second intervals; for most tests, a 30- to 60-second resolution is probably adequate. You need to log the data for at least 2 hours at a time. The length of time you let the baseline run isn't as important as *when* you let it run. The more closely your monitoring schedule approximates your real usage peaks, the more valuable the data will be—a 2-hour sample from Saturday doesn't give you as much information as a 2-hour sample from Monday. You should also do some sampling over longer periods to ensure that you're not missing any peaks that you don't know about.

In this data, you're looking for two things. One is peaks: you want to see where the high points of resource consumption are. The maximum value during your peak periods isn't that important. The real value comes from having the average value during peak periods. The Windows Performance Monitor tool can calculate the average value of two points on a performance graph, so you can use that feature to quickly identify the peak value of each of the counters listed earlier.

Monday mornings and Friday afternoons are probably peak times in your organization, as are the times immediately following lunch during the weekday. Depending on your business, you may have other peak times (such as the week before the close of a fiscal quarter or the time right before a major trade show or other event). Monitoring peak usage ensures that you can test the proposed configuration to determine what kind of performance it delivers under a simulation of your real peak loads. The second consideration is average resource usage; by checking this, you can view the normal workload that the server has to handle.

Environmental Testing

Exchange never operates in isolation; it depends on inbound and outbound DNS and AD access. It's a good idea to run tests on these related systems as part of your validation, testing, and deployment procedures. For example, suppose you're consolidating operations to a small number of regional data centers. When you do, what impact will the consolidation have on your global catalog (GC) servers? If you centralize GCs and domain controllers, what impact does that have on users' ability to log on in remote or branch offices? If you don't centralize the domain controllers, will there be an increased amount of replication traffic? Although the mechanics of doing these tests are outside the scope of this guide, the idea behind them is perfectly in line with the theme of this chapter—if you haven't tested it, it doesn't count.

Building a Test Plan

"Prior planning prevents poor performance." This saying is especially true of testing and validation, because in general, once you purchase hardware, you're stuck with it—if you only find out your configuration is flawed in some way after the purchase process is done, you're probably out of luck. The best way to ensure that your testing accurately reflects the capacity and performance of your proposed configuration is to create a detailed test plan that describes what you're going to test, how you're going to test it, and how you'll judge the results. The following list highlights some of the questions your test plan should attempt to address:

- *What explicit performance requirements exist*? These are often couched as SLAs, using language such as "inbound mail will be delivered to the recipient within 5 minutes of arrival" or "the mail system will be available from 8:30am to 6:00pm local time, 7 days a week and 365 days per year". Your test plan should list these requirements so that you can explain how you'll find out if the new system would meet them.

From the explicit requirements, you can often derive *implicit requirements*. If you have a 4-hour RTO, your servers can't hold more data than you can comfortably back up in half that time—so your explicit RTO requirement turns into an implicit limit on the size of your servers. Your derived requirements must be listed too, because you also have to show that your tests cover them. Common implicit requirements include RTOs and RPOs tied to specific business deadlines (such as the end of a trading day or the end of a fiscal quarter). Don't forget that these requirements can either be business-related or technical; you might well have an implicit requirement that limits the amount of mailbox data on a server so that the server data can be restored within your RTO.

- *What performance data do you already have*? The answer *should* be "detailed performance baselines," but if you don't have those, your test plan should include a detailed description of how you'll gather that data.
- *How will you test system performance?* Which tools are you going to use? How will they be configured? It's critical to keep track of the settings that you use during testing to ensure that your tests can be repeated if necessary. Use the baselining discussion presented earlier as a guide.
- *What are the acceptance metrics for the new system*? In other words, how will you decide whether the new system is successful? Your metrics might include factors such as "decrease the average user response time to less than 2 seconds" or "provide room for future growth to 3000 mailboxes." For each metric, of course, you also need to explain how you'll test that metric—for example, "decrease average user response time to less than 2 seconds as measured by the following LoadSim tests…"

- *Is anything different?* You may not be able to test the exact configuration you're planning to deploy; if so, make sure that your test plan describes where your test configuration differs from the configuration you plan to deploy.
- *Who's doing the testing?* It's great to get help from your hardware vendors, but don't let them do the tests for you, or you'll have no way to verify the numbers. I'm not suggesting that the vendors will be dishonest on purpose, but *you're* the one who best understands your requirements, so you should be in charge of running the tests.
- *What's not getting tested*? For example, suppose you're thinking about consolidating your regional Exchange servers to a smaller number of centrally located clustered mailbox servers, but that you're not able to touch the AD configuration because that belongs to another business unit. Your test plan should indicate that fact to make it clear which unknowns or dependencies may still exist.

Of course, the intent of the test plan isn't to ask these questions, but to answer them. A clear, well-organized test plan that describes in detail what you're going to test, how you're going to test it, and how you'll evaluate the results will make it much easier to determine whether a given set of hardware will do what you want it to do.

Success Through Breaking Things

Boeing has some amazing videos showing how they destructively test aircraft components; they bend, flex, stretch, and twist different parts until they shatter, crack, implode, or otherwise fail (usually in a spectacular way). These exercises serve an important purpose by testing aircraft parts under unusual conditions so that there are no in-flight surprises. You can do more or less the same thing as part of your test regime. You can start with something simple: during a LoadSim test, turn off or disconnect the primary GC that the Exchange server is using (you can find out which GC it's using from the General tab of the server properties dialog box). See what happens. Does Exchange find the next available GC properly? If so, how long does it take? If not, that tells you that something needs fixing in your environment before you move on to the deployment stage.

Of course, the network path between your Exchange servers and the rest of the world is another fertile area for experimentation. For example, what if your network switch fails (or gets unplugged)? If you're using redundant NICs or NIC teaming, does the connection fail over properly? What about the router that connects your Exchange servers' subnet to the rest of the network?

A few more technology-specific suggestions: if you're using clusters, wring them out by starting a LoadSim test and then failing the Exchange virtual server over to another node. Even better: immediately fail it back to the original node. If your proposed configuration includes replication, tamper with the replication process by interrupting network connectivity, overfilling the receive buffer on the remote end, or otherwise trying to make things break.

If you're deploying SANs, see what happens if you "accidentally" pull a fiber out of an HBA during a Jetstress test. Most sites that deploy SANs use multipathing, and this is a great way to test whether your vendor's multipathing solution works the way it's supposed to. The possibilities are limitless. Of course, you should be careful not to try these tests on valuable data in case you *do* find a previously unknown-to-you failure mode.

The Pilot Phase

It's often tempting to simply install things and try them out. Fortunately, there's a way to both satisfy that urge *and* meet the validation, testing, and deployment requirements explored in this chapter.

The goal of a pilot is to gain some experience with a system configuration by putting it into limited use, then applying what you learn to deployment of the "real" system. This process is familiar to those who have worked with enterprise software development, and it's become increasingly common for desktop OS and application deployment too. The same concepts apply to Exchange; you can deploy a new version or configuration of Exchange as a pilot project, see how it works, and use that knowledge to adjust your deployment plan. The steps to conducting a successful pilot are fairly straightforward:

- *Decide on a pilot scope*. What size pilot do you want to conduct? Maybe you want a single Exchange 2003 Server system so that you can experiment with its mobility and wireless access features. Perhaps you want to deploy it at each Internet entry point so that you can test the Intelligent Message Filter and Sender ID. It's important to define the scope first because that determines how much equipment you need, what software must be installed, and so on.
- *Decide on a pilot goal*. What do you want to learn from your pilot? Ideally, a well-integrated pilot will help you accomplish several things. First, it will provide real-world validation of your design and deployment plan so that you know that your design assumptions match real-world conditions in your environment. Second, it will help you identify which features or functionality you need to learn more about before putting them into production. Third, it will give you a means to identify what you need to teach users about the new system. This is particularly important if you're moving from Exchange 5.5, Lotus Notes, or Novell GroupWise.
- *Choose a pilot population*. How many users do you want to include, and where will they come from? It's important to get a good mix of users. Many organizations start their pilots with IT staff, which is a mixed blessing. The IT staff is used to balky technology, but performance or stability that's acceptable to an administrator might not be acceptable to line-of-business users.
- *Set exit criteria and assess risks*. When will your pilot officially be considered "done"? How will you know if you're ready to move to full production? What might go wrong, and what can you do about it? This step is all about answering these questions.

- *Perform the initial pilot deployment*. Microsoft calls this the “pre-pilot” phase. At this stage, you and your team should be the only pilot users so that you have an opportunity to work out any kinks that would prevent the full pilot from running normally, and it helps you identify any areas of user education that need attention.
- *Perform the full pilot deployment*. This step is where you put your pilot into full use in the test population. You could argue that this step should really be several substeps because during the actual pilot you should be gathering performance and usability data to help determine whether your pilot design will work when scaled up.
- *Run the pilot*. Part of the pilot scope is a determination of how long you’ll let the pilot run; once you reach your exit criteria, you’re ready to move from the pilot system to the *real* system. This, of course, requires you to have created a plan to move from the pilot system to the real one.
- *Review what you’ve learned*. The output of your pilot should be a list of adjustments to your validation, testing, and deployment plans. These may be simple (for example, you might decide that your heaviest users only consume about 0.7 IOPS instead of 1.0), or they may be complicated (maybe clustering isn’t right for your environment as evidenced by your pilot’s uptime). Regardless, you should be prepared to make adjustments to your validation, testing, and deployment process to incorporate the results of your pilot.
- *Proceed with your deployment*. At this point, you’re ready to move forward with your actual deployment—provided that you’ve taken the lessons learned from the previous steps into account and made changes to your deployment plan as appropriate.

There’s one additional meta-step that you may choose to take advantage of—you can create a transition plan to move from the pilot to your production system. The nature of this plan depends on the nature of your pilot. If your pilot is conducted by installing a small number of servers in what will become the production Exchange infrastructure, moving to production is probably a simple matter of adding more servers. If, however, the pilot’s conducted in an isolated environment, you’ll need to take into account the process of moving users’ mailboxes and accumulated data to the new production system.

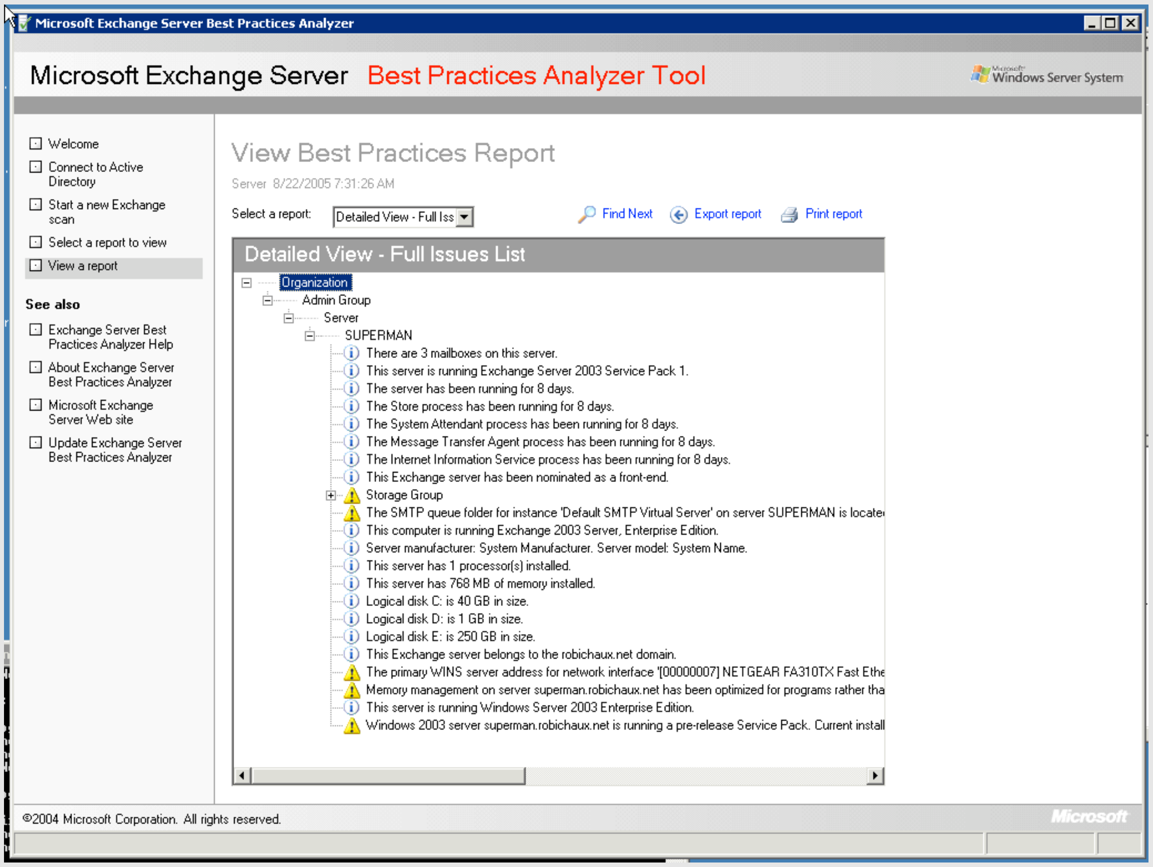
Deployment

Imagine that you’re done testing and you’re ready to deploy your new system. The next three chapters will talk about the mechanics of how you actually install and configure your disaster recovery, high-availability, and business continuity-related Exchange components and tools. Before you get to that point, though, you need to think about some broader issues related to your deployment, particularly how you’ll execute it. As with testing, the deployment phase of your project is most likely to go smoothly if you have a detailed plan that describes what you’ll do, how you’ll do it, and how you’ll know if your deployment was successful.

Ensuring a Smooth Deployment with ExBPA

If you want to ensure that your deployment goes smoothly, you should strongly consider running the Exchange Best Practices Analyzer (ExBPA) on your *existing* system. This exercise might seem pointless—after all, who cares what improvements could be made to the existing system when you're about to replace it? However, it's an idea that has a solid grounding in reality.

The whole idea behind ExBPA is to point out every configuration item in an Exchange organization that may be sub-optimal. Not everything that ExBPA finds is "wrong"; there are often good reasons to deviate from the rules that the Exchange product team has codified in ExBPA's analysis rule base. Running ExBPA against your existing system, though, might identify configuration problems that you haven't noticed, but which you wouldn't want to replicate in the new system. Figure 4.2 shows a sample report from ExBPA generated by scanning a test lab's front-end server.



Microsoft Exchange Server Best Practices Analyzer

Microsoft Exchange Server Best Practices Analyzer Tool

Windows Server System

- Welcome
- Connect to Active Directory
- Start a new Exchange scan
- Select a report to view
- View a report

See also

- Exchange Server Best Practices Analyzer Help
- About Exchange Server Best Practices Analyzer
- Microsoft Exchange Server Web site
- Update Exchange Server Best Practices Analyzer

View Best Practices Report

Server 8/22/2005 7:31:26 AM

Select a report: Detailed View - Full Iss

Find Next | Export report | Print report

Detailed View - Full Issues List

```
Organization
  Admin Group
    Server
      SUPERMAN
        There are 3 mailboxes on this server.
        This server is running Exchange Server 2003 Service Pack 1.
        The server has been running for 8 days.
        The Store process has been running for 8 days.
        The System Attendant process has been running for 8 days.
        The Message Transfer Agent process has been running for 8 days.
        The Internet Information Service process has been running for 8 days.
        This Exchange server has been nominated as a front-end.
        Storage Group
        The SMTP queue folder for instance 'Default SMTP Virtual Server' on server SUPERMAN is locate
        This computer is running Exchange 2003 Server, Enterprise Edition.
        Server manufacturer: System Manufacturer. Server model: System Name.
        This server has 1 processor(s) installed.
        This server has 768 MB of memory installed.
        Logical disk C: is 40 GB in size.
        Logical disk D: is 1 GB in size.
        Logical disk E: is 250 GB in size.
        This Exchange server belongs to the robichaux.net domain.
        The primary WINS server address for network interface '[00000007] NETGEAR FA310TX Fast Ethe
        Memory management on server superman.robichaux.net has been optimized for programs rather tha
        This server is running Windows Server 2003 Enterprise Edition.
        Windows 2003 server superman.robichaux.net is running a pre-release Service Pack. Current install
```

©2004 Microsoft Corporation. All rights reserved.

Microsoft

Figure 4.2: ExBPA in action.

You can see that the report includes information about uptime of the server, the system attendant, and the store process; it also points out that circular logging is turned off. Because this server is a front-end server with no mailboxes, it would be slightly more efficient to turn on circular logging—just the opposite of the situation on a mailbox server. ExBPA defines hundreds of individual checks such as these, ranging from serious to simple.

One exit criteria that I like to specify for the test phase is this: run ExBPA on the old system, and run it on the new configuration. If ExBPA reports any problems in *both* configurations, those problems either have to be fixed or signed off on by the responsible manager. This process provides a way to excuse legitimate deviations from ExBPA's recommendation while still guaranteeing that problems get fixed. Any problem reported in only the *new* system must go through the same process. Problems in the old system may or may not be fixed, depending on their nature and severity. Of course, ExBPA won't help you much with the actual deployment, but you can lean on the Exchange deployment tools (described in more depth in the next chapter) to help with that!

The first issue your deployment plan should cover is time—when are you starting, when are you finishing, and what intermediate milestones exist? Because the whole purpose of your deployment is to *improve* your Exchange availability, your schedule must take into account the cutover point when you'll switch to the new system. This point shouldn't be reached until after you've completed your acceptance tests, but it's important to identify the cutover point as early in the process as possible so that you have time to iron out any last-minute problems.

The second part of your deployment plan is the meat of the plan: the "what" and "how" of your new system. For something relatively simple (such as adding replication software to an existing configuration), this might only be a few paragraphs. For more complicated deployments, such as adding a SAN or consolidating to a clustered system, it will necessarily be longer and more detailed.

The next part of your deployment plan is the acceptance testing. What criteria will you use to decide whether the deployment was successful? Your criteria should include testing of basic functionality, such as sending and receiving messages. However, they must also include tests designed to prove that the new components meet the requirements you specified during the validation phase. For example, if one of your validation requirements is a 2-hour RTO, your deployment plan should include tests to verify whether that objective can be met. This process is subtly different from the tests you perform in the testing phase: those tests are designed to see whether the proposed configuration can meet your requirements; these tests are designed to prove that the configuration will do so in operational use. Your tests will probably include a mix of functional tests that check to make sure certain operations or processes work and performance tests that prove that your new system meets SLAs or other requirements.

Your deployment plan should also make provisions for gathering baseline performance data. This data is valuable because it lets you compare performance and resource consumption for the old and new systems and gives you a way to compare the initial system performance to its performance at a later date to see the effects of growth or reconfiguration.

Where to Learn More

Most administrators are used to doing ad-hoc validation, testing, and deployment, and the thought of facing a more structured environment can be a bit scary. Microsoft has a wide range of resources that you can use to help you determine which specific validation, testing, and deployment steps make sense in your environment, and how to carry them out.

- The *Exchange Server 2003 Deployment Guide* (http://www.microsoft.com/technet/prodtechnol/exchange/2003/library/depguide.mspx) is aimed at administrators who are deploying Exchange Server 2003 either for the first time or as an upgrade from older versions. From a validation, testing, and deployment perspective, it's full of useful step-by-step guidance about how to accomplish particular tasks that you'll need for all three validation, testing, and deployment phases.
- The documentation for LoadSim, Jetstress, and ESP are available from the Exchange tools page (http://www.microsoft.com/exchange/downloads/2003/default.mspx), along with a variety of other useful tools.
- The *Exchange Server 2003 Performance and Scalability Guide* (http://www.microsoft.com/technet/prodtechnol/exchange/2003/library/perfscalguide.mspx) has a wealth of information about performance measurement and benchmarking.
- Although it's slightly dated, Pierre Bijaoui's *Scaling Microsoft Exchange 2000* (HP Press; 2002) is a comprehensive reference to Exchange sizing and performance tuning. It's well worth a read as part of your efforts to plan your tests and test your plan.

Individual vendors such as Hewlett-Packard and Dell often have sizing and performance estimation tools on their Web sites; you can use these tools to double-check your assumptions about server sizing and SAN performance. When possible, it's probably a good idea to cross-check these tools against one another to smoke out any assumptions that the vendor has made about your configuration.

Summary

Planning and testing can be dry, dull work, and it's something most administrators would probably rather avoid. However, time spent in developing an accurate set of requirements, a comprehensive set of tests that determine whether the proposed configuration will meet those requirements, and a plan to put that configuration in place will be time well spent because it reduces the likelihood of buying the wrong thing, configuring it improperly, or causing other problems that lower, rather than raise, your availability and resiliency.