CLOUD ISSUE

Dr. Dobb's Journal

OCTOBER 2011

Cloud Databases

Selecting a database platform, associated tools, and middleware for deployment in the cloud

Next

ALSO INSIDE

Editorial >>

Getting Started with the Cloud: Amazon Web Services >>

From the Vault: Single Inheritance Classes in C >>

UBM TechWeb

www.drdobbs.com

Dr. Dobb's Journal

CONTENTS

October 2011

COVER STORY

7 Cloud Databases: Connectivity and Platform Options
By Ken North

Selecting a database platform, with its associated tools and middleware choices, for deployment in the cloud is a daunting task. Whether hosting a new database or moving existing databases to the cloud, you'll need to understand the present application architecture, and this article will help inform your decisions.

13 Getting Started with the Cloud: Amazon Web Services
By Andrew Glover

The ability to rapidly provision a wide range of computing resources on a pay-as-you-go basis has ushered in a new era of innovation; and the most widely used cloud platform is easy to configure and run, once you know how.

5 Editorial
By Andrew Binstock

19 From the Vault: Single Inheritance Classes in C
by Ron Kreymborg

Achieving C with Classes without resorting to C++ .

3 Letters
By you

Actors, certifications, and parallel data structures resonated with our readers.

29 Links

Snapshots of the most interesting items on drdobbs.com including User-Defined Hash Functions for Unordered Maps in C++0x and Running CUDA Code Natively on X86 processors.

30 Editorial and Business Contacts

More on DrDobbs.com

GData: Accessing Google-Application Data
Google's GData returns data in Atom or JSON formats. But the APIs are so poorly implemented, accessing the data is a project onto itself.
http://drdobbs.com/web-development/231600390

Improving Data Entry with UI Pickers in iOS
Cocoa's UI Pickers make it simple for users to enter data. But helping users avoid punching in invalid data gives the UX a magical feel. Here's how.
http://drdobbs.com/mobility/231600685

C++0x's Tools for Library Authors
Four additions to the language greatly facilitate writing C++ libraries.
http://drdobbs.com/cpp/231600433

Breaking Away From The Unit Test Group Think
Flooding a project with unit tests is rarely the right way to test it properly.
http://drdobbs.com/architecture-and-design/231600404

Jolt Awards for Books: The Rest of the Best
Given the closeness of the vote, we felt it would be worthwhile to present these other titles that made it to the final round of consideration — especially if they intersect with work you're currently doing.
http://drdobbs.com/joltawards/231600815

Member Function Pointers in D
Lambda functions are incredibly powerful, and for those not used to them, they continue to surprise and delight in what they can do in a simple, straightforward manner.
http://drdobbs.com/blogs/cpp/231600610

IN THIS ISSUE

Editorial >>
Cloud Databases >>
AWS >>
Single Inheritance >>
Letters >>
Links >>
Table of Contents >>

Mailbag

Actors, certifications, and parallel data structure dreams

Actors

In response to Parallel Microsoft-Style (http://drdobbs.com/architecture-and-design/231500268), many readers commented on their own experience of using actor-subsets published by Microsoft. Including:

I also wanted to mention that Microsoft has developed Parallel Patterns Library (PPL) in VS2010, which is similar to Intel's Threading Building Blocks (TBB) and is a very nice parallel library that uses the concept of tasks instead of threads. PPL also has many ready-to-use parallel patterns that make parallel programming much simpler, as TBB does as well.

— Victor J. Duvanenko

Andrew Binstock responds: "MS's efforts in this area are all over the map, it seems. From the comments on the editorial, there are four actor-like packages that the company has released or is thinking of releasing. It appears that MS is finally getting on board the bus!"

My team at Microsoft is in fact working on an actor-based parallel programming API for .NET, called "TPL Dataflow." Here is a link with background information: http://msdn.microsoft.com/en-us/devlabs/gg585582.

— iostrovsky980

Imagine WCF services consumed as in-process components. This capability is easily facilitated by WCF's powerful "self-hosting" architecture. Due to their very nature, these service-like "components" enjoy a formal isolation boundary that includes amongst a laundry list of other things, concurrency. And declarative concurrency to boot. What emerges is inherent Actor Model support. Microsoft doesn't even know what they've created. Co-option at its best.

As all the seasoned dogs know, the Smalltalk guys got many things right, including messaging as a fundamental aspect of architecture. Messaging is as loosely coupled as we can get.

— Michael Montgomery

IN THIS ISSUE

Editorial >>
Cloud Databases >>
AWS >>
Single Inheritance >>
Letters >>
Links >>
Table of Contents >>

Certifications

In your editorial, [Dude|Sir], That's Not Programming (http://drdobbs.com/joltawards/231500038), you stated:

"More accurately, there is a slight prejudice against candidates with many certifications. The more you have, the more suspect you become. They are like reverse merit badges."

Our manager where I work encourages us to get at least one certificate annually, with the company paying for the materials costs. I had never heard anyone say anything negative about someone having certificates. Can you shed more light on this? Is this something frowned upon in the industry?

— Scott Rabon

Andrew Binstock replies: "If your certification is for your own knowledge, there's nothing wrong with them at all. However, as a gauge of a developer's quality, they don't indicate much other than the ability to study and pass an exam. The more certifications you have, the more recruiters view this test-taking as being your primary skill. They will tend not to view you as expert in as many areas as you have certificates — rather, you appear to them as broad, when in fact most places are looking for deep.

There is also a sense that certification is a bit of a money grab. Training is expensive, study materials are expensive, and the tests themselves are expensive. And everyone in the food chain has a stake in candidates passing. Consequently, the cert gives you entry into a fraternity in which lots of other players play, rather than into a restricted club of established talent.

You can see more of thoughts on this by search for "certification" on reddit.com/programming where this topic comes up often and rarely gets full-throated endorsement, even from folks with certifications.

Parallel Data Structures

In response to Clay Breshears' inaugural blog on *Dr. Dobb's Go Parallel* (http://drdobbs.com/go-parallel/blogs/architecture-and-design/231601211), where he pondered whether there were any truly parallel data structures:

Data structures are basically structured collections of data items. This implies two components — the data items, and the structural information (e.g., simple linked list — data items plus pointers to next item). Concurrent access to individual data items is a well-studied problem. If you want safety, you can't get around having some mechanism to enforce atomic reads and writes, which implies some sort of locking scheme. However, applied to single data items, this is not likely to be a major bottleneck in most cases.

The real problem is the structural information — it has to be coordinated across the whole data structure or the whole structure can break down (for existing data structures, anyway). That means locking the whole data structure for any operation that will change the structural data, and that's likely to be a huge bottleneck. The only way around it would be a data structure where the structural information associated with each item could be modified based solely on local information, without breaking the structure. I can't imagine how to create such a data structure, but that's what would be required for a truly parallel data structure.

—George Rappolt

Have a correction or a thoughtful opinion on *Dr. Dobb's* content? Let us know! Write to Andrew Binstock at alb@drdobbs.com. Letters chosen for publication may be edited for clarity and brevity. All letters become property of *Dr. Dobb's*.

IN THIS ISSUE

Editorial >>
Cloud Databases >>
AWS >>
Single Inheritance >>
Letters >>
Links >>
Table of Contents >>

Who Are You?

Knowing your biases and preferences makes you a much better team-member

One of the most common, but least compelling, metaphors is the three-legged stool. It's trotted out anytime a speaker wants to show that three core values or activities enable some desirable goal. It implies that if any one leg should falter, the whole thing falls apart. This image has never worked for me because speakers generally gloss over a key aspect: For the stool to be useful at all, three legs must advance at the same rate. Which, of course, is frequently not a goal or even realistic. Moreover, most projects cannot be boiled down to just three essential aspects.

In a programming context, the latter point is often articulated as: "Cheap, fast, good — choose any two." (Here "fast" refers to delivery of product, not its inherent performance.) This little quip is often used as an adage in meetings with naive managers who want all three things from their developers. On further examination, however, it's clear that the dynamic is really between only two poles: quality and everything else. Consider the possible option pair of fast and good. Is cost really the principal obstacle to that? We know from Fred Brooks (http://is.gd/RU3eCx) that spending more on people doesn't help the speed-quality combination. And so what cost is it that prevents fast and good? There is none. The three dimensions simply aren't related by opposing tensions.

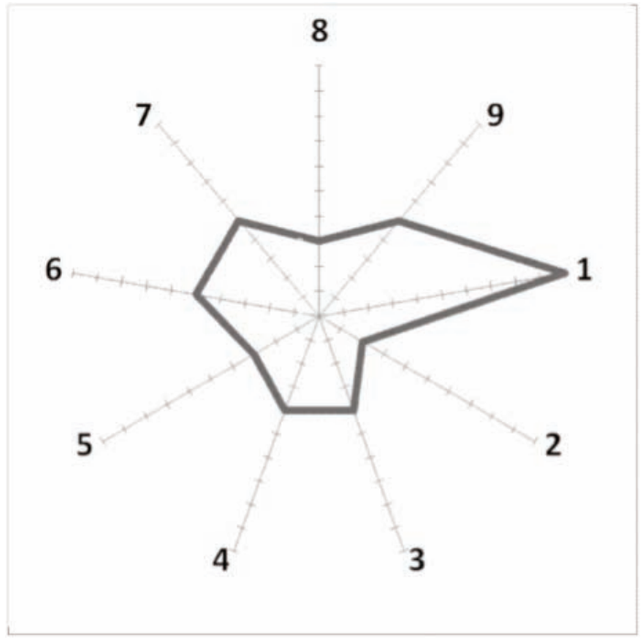
Cheap-fast-good is only one possible set of values. Additional values are scalable, performant, secure. You can add many others, as long as they don't overlap. (Although one could well argue that scalability and performance are overlapping, possibly even congruent, features.) Eventually, you end up with a radar chart, such as the one shown on the next page, of the qualities and their respective value to a project.

With little effort, it's possible to draw a similar chart for the organization in which you work. Fairly often, the comparative importance of the key qualities are well known and intuitively understood. (Performance is always crucial, cost is a mid-level concern, security is low level, etc.) So it's not difficult to map them to a radar chart. In a world where everything was properly documented, such a chart would be given to every new member of the team, so that they would know the internal values that drive decisions.

This diagram becomes far more valuable when taken to its logical extension: Plotting this graph for yourself. What do *you* value in program-

IN THIS ISSUE

Editorial >>
Cloud Databases >>
AWS >>
Single Inheritance >>
Letters >>
Links >>
Table of Contents >>



ming? Like most of you, I've met all kinds of developers. Those who obsess over performance and will write unmaintainable code to achieve it. And there are those who are very careful coders, concerned with correctness, and not much worried about performance. Others are obsessed with getting code out the door and are willing to accept small bugs in in-house apps to clear the backlog. The important thing in this self-assessment is to describe what you value (rather than what you're good at).

The resulting chart can show you that, in fact, you don't belong in the organization you're currently working in. You might know this intuitively, but now you can now see why: Your interests are not rewarded by the organization's values. On the other hand, if you're happy where you are, it can show you what skills you need to do to be a better contributor (as well as what pointless conversations to stop having.)

In an upcoming editorial, I'll discuss my belief that a developer's profile tends to dictate the use of certain tools and techniques, often conglomerated in ways that are not ideal because they represent values that are frequently not self-consistent. 'Til then...

— *Andrew Binstock is Editor in Chief for* **Dr. Dobb's** *and can be contacted at alb@drdobbs.com.*

Comment

Cloud Databases: Connectivity and Platform Options

If you plan to host databases in the cloud, you should be attuned to platform differences

By Ken North

An examination of the database landscape reveals that, along with social networks, big data and cloud computing have been game changers. Those responsible for existing databases, and developing new applications or services, will prudently look at the public, private, and hybrid cloud options for hosting big data and databases.

Building a new application means developers have great flexibility in choosing what architecture, network, and database platform will match requirements. But migration of an existing database to the cloud means developers must understand how the cloud database will support existing applications, with the goal being transparency and a seamless transition. Ideally, migration to the cloud should not require expensive reprogramming of applications and services.

If you are planning to host databases in the cloud, you should be attuned to platform differences, such as data model, language bindings, service-level agreements, replication, and tools for administrators and developers. You should also examine issues related to operating systems, hypervisors, authorization, authentication, and connectivity — just to name a few.

If you're moving existing databases to the cloud, you'll need to understand the present database and application architecture, APIs, libraries, and uptime requirements. There can be SLA, availability and licensing issues for middleware, application servers, and database servers.

This discussion is a look at some platform and connectivity options. For this purpose, think of connectivity as encompassing network protocols and data access application programming interfaces (APIs).

Connectivity: APIs and Protocols

The ascendancy of the SQL database was due in part to distributed processing based on client-server architecture. Clients ship queries over a network for processing by the database server. In this context, "client" refers to a database server client, which could be an application or service, often running on an application server or web server.

"The REST and XML solution has seen widespread adoption for web applications. Besides XML, a simple data interchange format, JavaScript Object Notation (JSON), has also gained traction in recent years for web work"

Network architectures, such as IBM Distributed Relational Database Architecture (DRDA) and Oracle Transparent Network Substrate (TNS) define specific protocols ('wire protocols') for communicating with the database server. Sybase Adaptive Server and Microsoft SQL Server operate with their own version of the Tabular Data Stream (TDS) wire protocol.

Because database wire protocols sit at a higher level in the network stack, they can operate over lower level protocols, such as TCP/IP connecting to database servers in the cloud, data centers, stores, and offices.

The client software stack for an SQL database includes libraries to implement a database access API and a network library for the platform-specific wire protocol. There are also JDBC and ODBC drivers that implement the wire protocol and therefore don't require a network library on the client.

The W3C XML specification provided a vehicle for building a new class of highly interoperable messaging capabilities, based on protocols that exchange XML-encoded messages. XML messaging provided a solution for web services using SOAP and REST interfaces. They also enabled developers to create data services that could integrate information from disparate data sources, SQL and non-SQL, and deliver it to the browser and other clients.

The REST and XML solution has seen widespread adoption for web applications. Besides XML, a simple data interchange format, JavaScript Object Notation (JSON), has also gained traction in recent years for web work.

Companies with SQL platforms, such as Microsoft, and those with non-SQL data, such as Google, have adopted data delivery protocols based on XML and JSON data sent by `HTTP:` commands.

AtomPub

The Atom Publishing Protocol (AtomPub), RFC 5023, uses XML and HTTP methods (`GET`, `POST`, `PUT`, and `DELETE`) to discover and operate with resources and collections of resources. It uses Atom-formatted data to describe the state and metadata of resources. The JavaScript Object Notation (JSON), RFC 4627, is a data interchange format consisting of structured data represented as text. JSON is based on two data structures: an ordered list of values and a collection of name/value pairs.

A variety of Google products expose APIs based on the Google Data Protocol (GData), which operates with data feeds that are in AtomPub

and JSON format. Developers can program directly with GData using HTTP `GET` and `POST` requests, or they can use instead client libraries that handle the HTTP `GET` and `POST` processing. To reduce network round trips, GData supports batching multiple operations in a single `POST` request.

The Open Data Protocol (OData) was developed at Microsoft before being released as an open specification. OData supports URIs as resource identifiers, HTTP, and operations with AtomPub and JSON data feeds. OData recently added support for geospatial data types as a set of primitives. There are a variety of OData providers and libraries for .NET Framework, Java, Rails development and support for Objective-C, JavaScript, and PHP. OData.org operates an OData service validation tool at http://services.odata.org/validation/.

Cloud Databases: Platforms, Engines, and Data Stores

The cloud database landscape today is marked by diversity. In that respect it is not unlike the database landscape 20 years ago. Back then there were products that ran on the desktop, on file servers, on mainframes, and on SQL servers. Today we have all of those database types but we also have powerful database engines embedded in mobile devices and client-server databases hosted in the cloud. The cloud database community is experiencing the cornucopia of APIs effect that the SQL database community experienced long ago. New cloud database platforms arrived in recent years, bringing with them disparate programming interfaces for data access.

We're seeing thousands of Memcached and NoSQL deployments. NewSQL platforms are emerging and the largest database vendors (IBM, Microsoft, Oracle) are carving out their share of the cloud computing market.

Moving to the cloud does not require rewriting application code. Organizations can port enterprise applications and databases to the cloud, such as by using Amazon EC2 Relational Database AMIs to operate with EnterpriseDB, IBM DB2, MySQL, Microsoft SQL Server, Oracle Database 11g, PostgreSQL, Sybase IQ and SQL Anywhere, and Vertica. This strategy moves the data to the cloud without having to reprogram a data access layer that uses ADO.NET, ODBC, JDBC, or proprietary APIs.

Organizations can transport enterprise applications and databases to the cloud while retaining ODBC, JDBC, .NET, and Java data access code. Likewise they can choose to host parts of a service-oriented architecture (SOA) in the cloud, preserving APIs and the logic of data services and XML-based web services.

Developers creating new applications for the cloud — not migrating existing code and databases — can use the aforementioned platforms or look to other cloud database alternatives. There are too many choices for one article, so we'll look at selected platforms here.

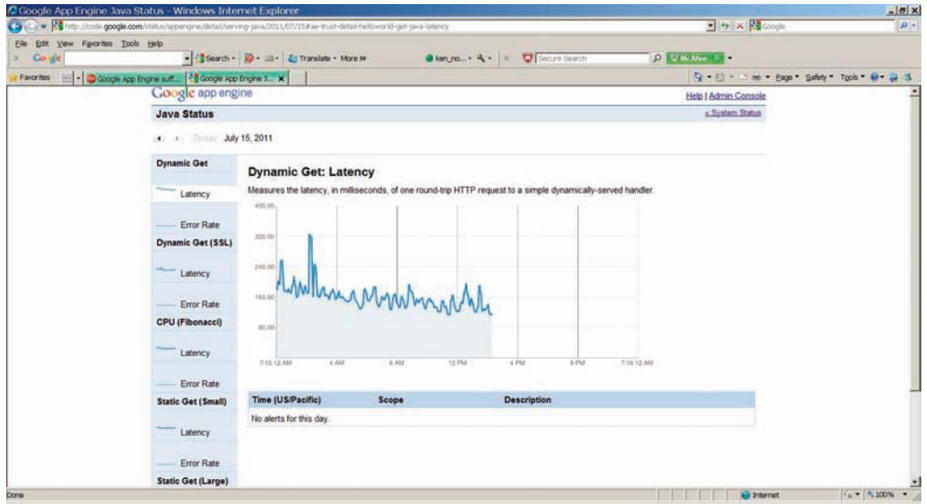


Figure 1: Google App Engine Status Display.

Google App Engine

Developers using Google App Engine have a choice of APIs for data storage, using the built-in Datastore and High Replication Datastore (HRD) for Python and Java development. Besides APIs such as Mail, Images, and Taskqueue, App Engine developers can operate on in-memory data with Memcache. The Datastore Java API supports Java Persistence (JPA) and queries using Java Data Objects (JDO). The HRD provides the advantage of replicating data across data centers using a solution based on the Paxos algorithm. There is also a master/slave replication alternative that supports asynchronous replication across data centers.

Developers can also check the status of Java and Python services using the Google App Engine status dashboard.

Google has committed to supporting hosted SQL, full-text search, and MapReduce in upcoming releases of App Engine. All paid users will have a 99.95% uptime service-level agreement, but HRD has achieved 99.999% uptime since its launch.

Amazon Relational Database Service (RDS)

Amazon EC2 developers can use preconfigured AMIs for the database products discussed above or opt for the Amazon Relational Database Service (RDS) to operate with Oracle and MySQL databases. RDS is a web service that provides pay-per-use capacity and freedom from the overhead of database administration, such as the backups, replication, and the arduous task of keeping up with security patches. With RDS, developers are able to use familiar tools and administrators can use Amazon CloudWatch to monitor storage and computer resource consumption. A principal advantage of RDS is the scalability and elasticity of the cloud, being able to bring resources to bear when you need them.

Amazon RDS for MySQL supports read replicas and deployment across multiple Availability Zones. Amazon RDS for Oracle provides automatic host replacement in the event of a computer instance failure, but it does not currently support replication. Amazon RDS for Oracle also supports provisioned database storage. Each DB Instance can select from 5GB to 1TB for its primary data set at a rate between $0.10-0.12 per gigabyte per month.

Microsoft SQL Azure

Microsoft SQL Azure is a pay-as-you-go solution that enables developers familiar with SQL Server to use familiar tools when they move to a cloud database. SQL Azure integrates with Visual Studio and SQL Server and supports Transact-SQL (T-SQL) and the TDS protocol. Like Database.com, it's built with a multi-tenant architecture, meaning that

other users are sharing the database storage. For data access, SQL Azure supports ODBC, ADO.NET, JDBC, and PHP. SQL Azure also provides data synchronization capabilities based on Microsoft Sync Framework technologies. This permits data replication and synchronization within and across data center boundaries.

Developers can create OData feeds from SQL Azure or Microsoft SQL Server. OData, formerly known as ADO.NET, operates with technologies such as HTTP, AtomPub, and JSON, to deliver information from applications and services.

ParAccel

Like Sybase IQ and Vertica, ParAccel is a columnar database. That means its data model is optimal for analytics, not transaction processing. It can operate as a disk-based or in-memory database. According to ParAccel CEO Barry Zane, the sweet spot for the in-memory database is 2 terabytes (TB), whereas it's 25 TB for an on-disk database.

ParAccel offers multiple programming interfaces, including an SQL API and Map/Reduce.

Xeround MySQL Cloud Database

Xeround's claim to fame is tailoring MySQL for the cloud environment. The Xeround offering is a distributed database that runs in memory across multiple nodes. It's suitable for transactional applications, not for analytics and business intelligence (BI). One notable aspect of Xeround is it is cloud-agnostic, running on Rackspace, Heroku, and Amazon Web Services (AWS). The sweet spot for Xeround's performance advantages is databases whose size ranges from 2 gigabytes (GB) to 50 GB. Xeround supports auto scaling after you set CPU utilization, memory utilization, and connection upper and lower thresholds.

Xeround has published online transaction processing (OLTP) benchmark results that compare Amazon RDS MySQL performance with that of Xeround Cloud Database, for 1 to 240 concurrent users.

DBT-2

DBT-2 is a transaction processing benchmark, similar to TPC-C, with the source code available at SourceForge (http://is.gd/bPHpxg).

DBT-2 simulates a parts supplier with several workers accessing a database, updating customer information and checking on parts inventories. The test metrics include transactions per second, CPU utilization, I/O activity, and memory utilization. For 60-240 users, Xeround sustained 8000 transactions per minute whereas RDS performance peaked at 15 users and declined between 30-240 users.

Cloudant

Cloudant has emerged as a platform that operates with semi-structured data in a highly distributed, scalable architecture. Although it is NoSQL database technology, CEO Mike Miller has described Cloudant as relationally complete. Cloudant is available as a download or as a cloud database on Amazon EC2. Hosting a large multi-tenant architecture on EC2 proved to be a challenge.

Cloudant users communicate with the database from the web browser, using the HTTP: protocol. Cloudant feels this enables a client to talk directly to the database without going through a middle-tier server. But in my mind this raises questions about the type of application for which Cloudant is appropriate. Without middle-tier servers, application-specific logic such as business rules must be handled in the client or in the database. Since the client is the web browser, that means having to create sophisticated HTML, scripts, and browser add-ins.

The size sweet spot for optimal Cloudant performance is 1 GB to 100 TB. Cloudant has recently introduced a solution that bundles Apache Lucene plus CouchDB, an integration that supports full-text search and real-time analytics.

NuoDB

NuoDB, formerly NimbusDB, is perhaps best-known because its chief architect is industry icon Jim Starkey. Starkey has long been involved in creating database management systems and is the father of multi-version concurrency control (MVCC). NuoDB represents the NewSQL approach of supporting scalability with a radical shift of architecture; it offers support for standard SQL grammar and APIs. NuoDB provides the capability to scale out horizontally without giving up support transactions having atomicity, consistency, isolation, and durability (ACID) properties.

NuoDB is not generally available yet so there's no empirical data about optimal database sizes. The founders will point out that it's intended more for transactional applications than for big data applications, such as analytics with very large data sets.

Database.com

Cloud computing has IT organizations thinking about the advantages of computing as an expense that does not require large capital expenses for infrastructure. This tread is behind the interest in database-as-a-Service, such as the Database.com service from Salesforce.com. Developers writing apps for the Salesforce CRM service or for the Force.com platform use a database structure built with a multi-tenant architecture. Because a database can include data from a multitude of Salesforce customers, the architecture provides a very sophisticated security model. Besides authentication and authorization capabilities (found with a robust DBMS), Database.com supports encryption, roles, profiles, sharing rules, and privileges tied to user ID and session. In addition, the validity of database requests can be checked against access hours and originating IP address.

Database.com is an SQL database solution. Clients are able to access data using JDBC drivers, ODBC drivers, and REST and SOAP web services interfaces. Developers are able to use a variety of languages and tools, including Java, PHP, Ruby, .NET, Adobe AIR, and Apex.

Not a Slam Dunk

Selecting a database platform, with its associated tools and middleware choices, for deployment in the cloud is a daunting task. The choice is a bit easier when the job is to move an existing database into the cloud, but there are still configuration issues to resolve, such as whether to fail over to a different EC2 availability zone. Not even a move into a private cloud is guaranteed to be a slam dunk due to infrastructure issues, such as storage options, operating systems, and hypervisors.

— Ken North is a well-known expert in database technologies and regular contributor to **Dr. Dobb's**. *Read his blog at http://drdobbs.com/author/6831.*

Comment

IN THIS ISSUE

Editorial >>
Cloud Databases >>
AWS >>
Single Inheritance >>
Letters >>
Links >>
Table of Contents >>

Getting Started with the Cloud: Amazon Web Services

The most widely used cloud platform is easy to configure and run, once you know how

By Andrew Glover

Amazon Web Services, or AWS for short, is a suite of cloud-based, pay-as-you-go, on-demand services that facilitate building web applications by providing the primary infrastructure components. With AWS, you can rapidly provision (and scale as needed) computing resources, storage, and even messaging. In fact, AWS supports the development and consequent deployment of entire web architectures — ranging from the actual hardware infrastructure for entire web applications to reside to content delivery networks. In this series of articles, I will explain how to set up your AWS instance (this article), move your apps to it and run them there, and how to scale up resources as you need. As expected, the programming aspects of this will be a core focus.

AWS is a game changer. Anyone — hobbyists to large IT departments, — can bring full-fledged web applications into production at a pace and price not seen before. The capital-intensive process of building an infrastructure in anticipation of load is dead. Entire working products (that is, not the prototypes or small pieces, as in decades past) can be rapidly deployed for less money than it costs to acquire a few computers.

Getting Started

Climbing on board AWS is comparatively easy; however, as you'll find out quickly, AWS is not free. While there are free usage tiers (i.e., you can use lightweight EC2 instances at no cost per hour), all AWS products are provided on a pay-as-you-go basis. You pay for what you use, whether that be computing resources, bandwidth, or storage. Thus, in order to begin using Amazon's suite of tools, you must create an account with Amazon and provide a credit card.

If you are like me, you've come to expect just about everything related to building software for free. But for the moment, don't let the thought of spending money stop you from exploring AWS as its pricing is low, especially if you plan to do some prototyping in the beginning. For less than the cost of a cup of coffee, you can deploy a working application that leverages multiple AWS products.

To create an AWS account, simply go to http://aws.amazon.com — of key importance throughout this entire process is the creation of your account's various credentials. While you'll have sign-in credentials (that is, a username and password), which you'll use to sign into Amazon's

management console, you'll also need access credentials. You will use them to accomplish various things on AWS.

Once you have an account, there several ways for interacting with AWS: the AWS Management Console, the command line, or programmatically through an API. There are standard libraries offered by AWS for Java, Ruby, Python, and .NET, just to name a few. What's more, because the AWS API is a series of web service calls, there are alternative open source libraries (in a wide variety of languages including C++) for working with AWS.

I think you'll find that in some cases working with the AWS management console is much easier, while in other cases we'll discuss later, using a programmatic API or command-line tools provides better flexibility when it comes to automation.

AWS is made up of several products. The touchstone of the entire suite is the Elastic Compute Cloud (or EC2). EC2 is essentially a virtual computer running the operating system of your choice along with various options for memory, CPU speed, and storage. With EC2, for example, you can quickly provision a 32-bit Linux image running any one of a menu of Ubuntu versions with 1.7 GB of memory and, say, 160 GB of storage.

Thus, EC2 is the basis of an application's infrastructure, on which other software assets are deployed. This is different from other cloud applications deployment options, such as Google's App Engine, Heroku, or event AWS's own Elastic Beanstalk. While all of those products allow you to deploy web applications, they do so by providing a platform, which handles the minutia of memory requirements, file systems, and the like. In fact, the difference between the rawness of EC2 and the bundled nature of something like Google's App Engine has lead to two terms: Infrastructure as a Service (IaaS) for the EC2 model and Platform as a Service (PaaS) for Google App Engine, Heroku, and others.

EC2 is pure infrastructure: It's bare bones computing power. Everything else in your instance is handled by you. In fact, provisioning an EC2 instance is not all that different from going out and buying a server with Linux on it; the biggest difference between the two is the lack of an upfront cost for EC2.

Before beginning the process of provisioning an EC2 instance, you need to work out two aspects:

- Your choice of operating system and a corresponding Amazon Machine Image (or AMI)
- Your memory, space, and CPU requirements

With EC2, you can choose from Windows, OpenSolaris, and a variety of Linux distributions ranging from Fedora to Ubuntu, just to name a few. I tend to prefer Ubuntu and there are several freely available images from the Ubuntu team to choose from. These OS images are known as AMIs. Some are freely available and others require payment; what's more, there are AMIs that come loaded with software packages (for example, an Enterprise Oracle Linux AMI running an Oracle DB).

There are various classes of EC2 instance types on which you run AMIs ranging from the low end all the way up to the high end. There are also instance types within classes — for example, High-Memory instances with 17.1 GB of memory all the way up to 68.4 GB.

Remember, with EC2, you pay for what you use, and the price per resource varies depending on the resource type. For instance, at the entry level, a small instance (within the standard instance class) is a single core having 1.7 GB of memory on a 32-bit platform with 160 GB of local storage. It will cost you $0.085/hr if you go the Linux route or about $2.00 a day. But, if you fire up a High-Memory Quadruple Extra Large

IN THIS ISSUE

Editorial >>
Cloud Databases >>
AWS >>
Single Inheritance >>
Letters >>
Links >>
Table of Contents >>

instance possessing 68.4 GB of memory, 8 cores, 1690 GB of storage running Linux, it will run you $2.00 an hour. Windows costs slightly more in each case. Keep in mind, however, that the price I just quoted is pure computing cost — there could be additional charges depending on how much bandwidth you utilize, etc.

Once you've settled on an instance type and OS, you need to do two more things:

- Set up a private key pair
- Determine a security policy

Provisioning an EC2 Instance

EC2 images are bare bones propositions, so you have to provision them — that is, install whatever software you want running on them. Thus, you'll need to log into the EC2 images via SSH. And to do so, you'll need an EC2 private key pair created before you fire up an image. The easiest way to get the keys is to create them via the AWS management console.

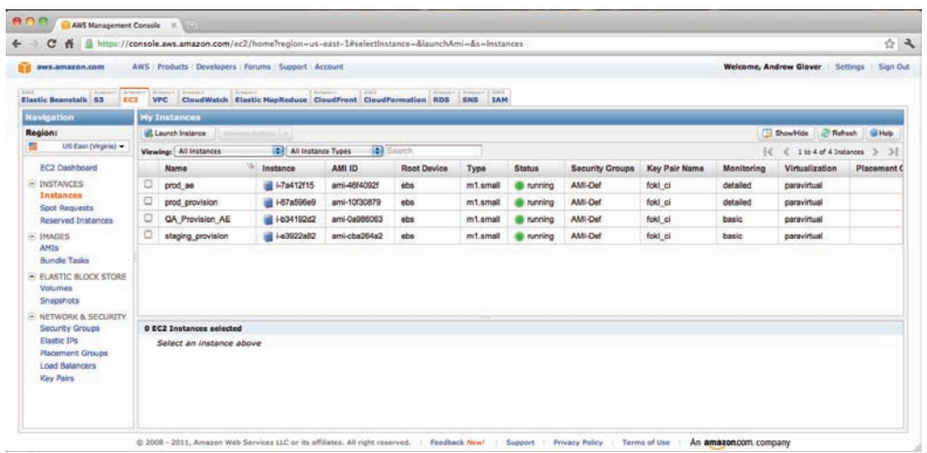


Figure 1: The AWS management console.

For example, once you've signed into the AWS Management console, select the EC2 tab. From there, you should see a Key Pairs link in the left-hand navigation bar, as shown in Figure 1.

If you click the Key Pairs link, you will be presented with a view containing a button labeled Create Key Pair, which if you click, will allow you to name a key pair and consequently download it.

Alternatively, you can also create a key pair using AWS's command-line tools. Configuring these tools is fairly simple, provided you're comfortable with the command line and altering your path. Once you've installed these tools, you can create a key pair effortlessly; like so:

```
$>ec2-create-keypair my_new_keypair
```

This script will output an RSA private key, which you'll need to copy and paste into a file — in this case, I'd call the file `my_new_keypair.pem`; don't forget to run chmod 600 on the file, so that it is indeed kept private.

Next, you'll need to create a security policy for your running image. In EC2 terms, this means specifying what ports are open to traffic. For the purposes of this article, I'll be enabling SSH and HTTP/HTTPS. If you need other services, like FTP or non-standard ports for HTTP (like 8080) open to the outside world, you'll need to explicitly enable those ports.

To define a security group, click the Security Groups link in the left-hand navigation bar of the EC2 tab. As with key pairs, there will be a button labeled Create Security Group, as in Figure 2. Click it and you'll be asked to name your security group; what's more, you can then create rules for inbound traffic.

IN THIS ISSUE

Editorial >>
Cloud Databases >>
AWS >>
Single Inheritance >>
Letters >>
Links >>
Table of Contents >>

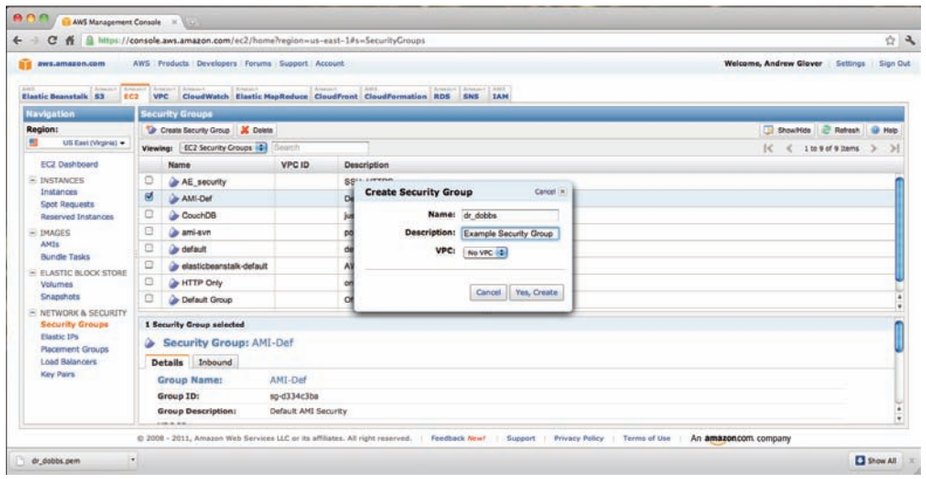


Figure 2: Creating a security group.

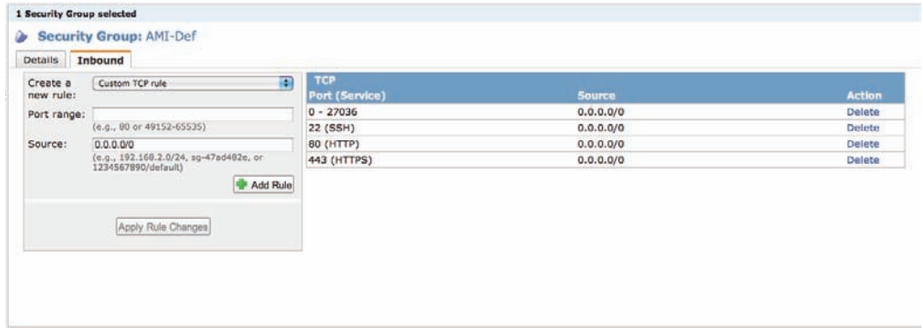


Figure 3: Security group example.

Security group rules are a bit confusing at first, but they are simple to define. For instance, a source of `0.0.0.0/0` means from any outside request, which is probably what you want for now. You can see though that you can fine tune your security policies using this scheme (Figure 3).

Once you've defined a security policy and created a private key pair, you are ready to fire up an EC2 AMI in one of two ways. You can programmatically fire one up or you can start an EC2 AMI via the AWS management console.

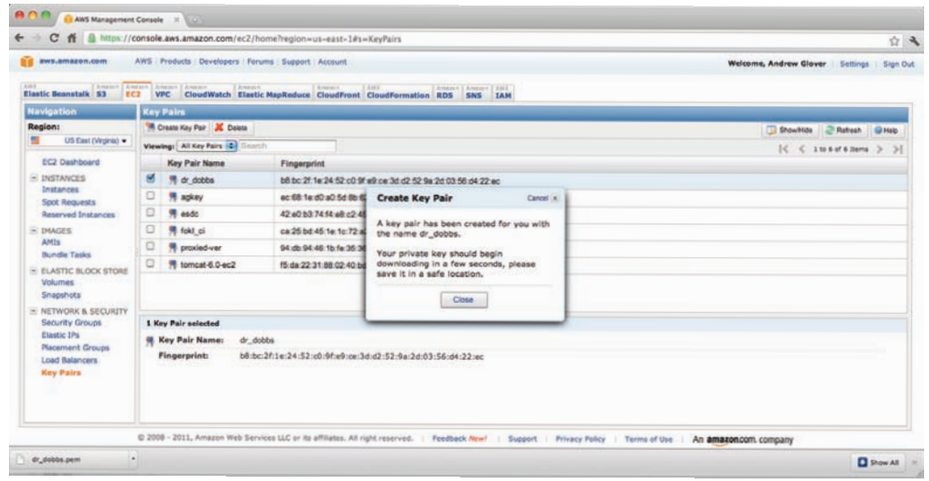


Figure 4: Creating a key pair.

As I mentioned earlier, there are several languages to choose from when it comes to programmatically interacting with AWS and you are free to use AWS provided SDKs or open-source independent implementations.

Using the AWS Java SDK, for instance, is fairly easy to start an AMI — all you need to do is tell AWS which AMI you'd like to start, what type of instance you'd like to provision, what security policy to apply, and finally what private key pair (Figure 4) to use:

```
AmazonEC2 ec2 =
  new AmazonEC2Client(new BasicAWSCredentials("....", "..."));

  RunInstancesRequest runInstancesRequest = new RunInstancesRequest()
          .withInstanceType("t1.micro")
          .withImageId("ami-46f4092f")
          .withMinCount(1)
          .withMaxCount(1)
          .withSecurityGroupIds("AMI-Def")
          .withKeyName("my_key");

  RunInstancesResult runInstances = ec2.runInstances(runInstancesRequest);
```

IN THIS ISSUE

Editorial >>
Cloud Databases >>
AWS >>
Single Inheritance >>
Letters >>
Links >>
Table of Contents >>

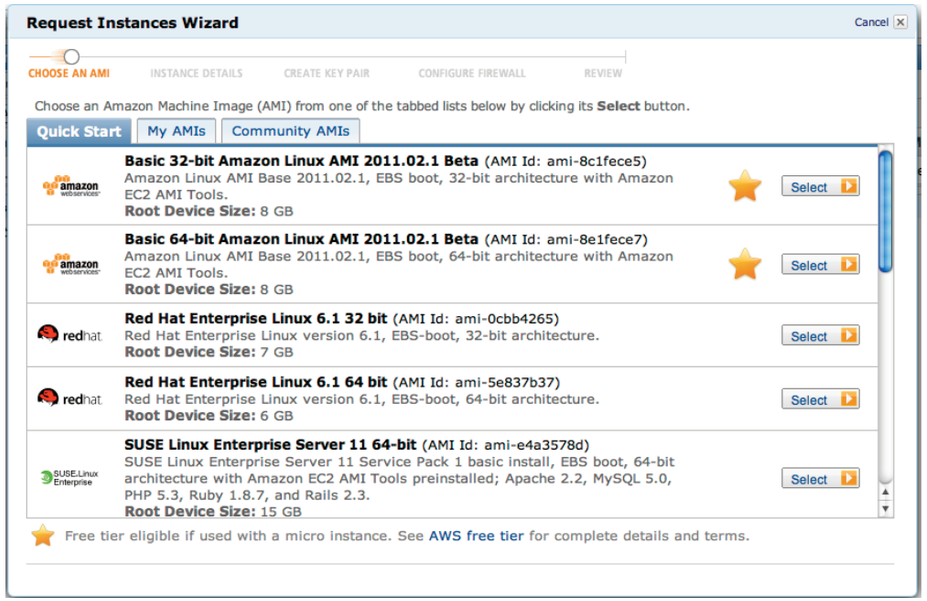


Figure 5: Launching an instance.

Alternatively, you can sign into the AWS management console, and within a few clicks, launch an AMI.

The provided wizard makes it very simple to select an AMI, an instance type, and you're free to select existing key pairs or even create them then and there.

For instance, from the Instances link on the left navigation bar, you can click the Launch Instance button, which starts a handy wizard (Figure 5).

Finally, you can also provision an instance via the command line — using the `ec2-run-instances` command, you can specify an AMI, key pair via the `-k` flag, and security group via the `-g` flag, to name a few of the available options.

```
ec2-run-instances ami-46f4092f -k dr_dbs_pair2 -g AMI-Def
```

It should be noted that with the command line, you can start an EC2 instance without any security group (in which case the default group will be added, which is to say no port is open!). You later add authorized ports via the `ec2-authorize` command.

The EC2 AMI that I tend to favor these days is `ami-46f4092f`, which is an Ubuntu image provided by the Ubuntu team. Ubuntu has quite a few official AMIs to choose from running various versions of Ubuntu as well as AMIs for i386 or AMD64 architectures.

Of course, firing up an AMI on EC2 is the easy part —making use of it is another story. When you provision a Linux AMI with EC2, like `ami-46f4092f`, you can SSH to it using the key pair you assigned to that instance. To do so, you tell SSH which key to use:

```
ssh -i .ec2/dr_dobbs.pem ubuntu@ec2-xxx-xx-xx-xxx.compute-12.amazon-aws.com
```

Most AMIs provide instructions on whom to SSH as, for instance, Ubuntu images use the `ubuntu` user, who has `sudo` privileges. Of course, once you're on your EC2 instance, you need to do something with it. In this case, I'm going to install a few packages —namely, Java 1.6, Git, and a few system utilities.

Installing core libraries and platforms, such as Java or Ruby on an Ubuntu image can be somewhat of a pain; consequently, I ended up creating a series of scripts that make this process pretty simple. The

project containing the scripts is hosted on Github and is dubbed `ubuntu-equip`. To install Java, for example, once you are on your desired EC2 instance via SSH, type

```
wget --no-check-certificate https://github.com/aglover/ubuntu-equip/raw/master/equip_java.sh && bash equip_java.sh
```

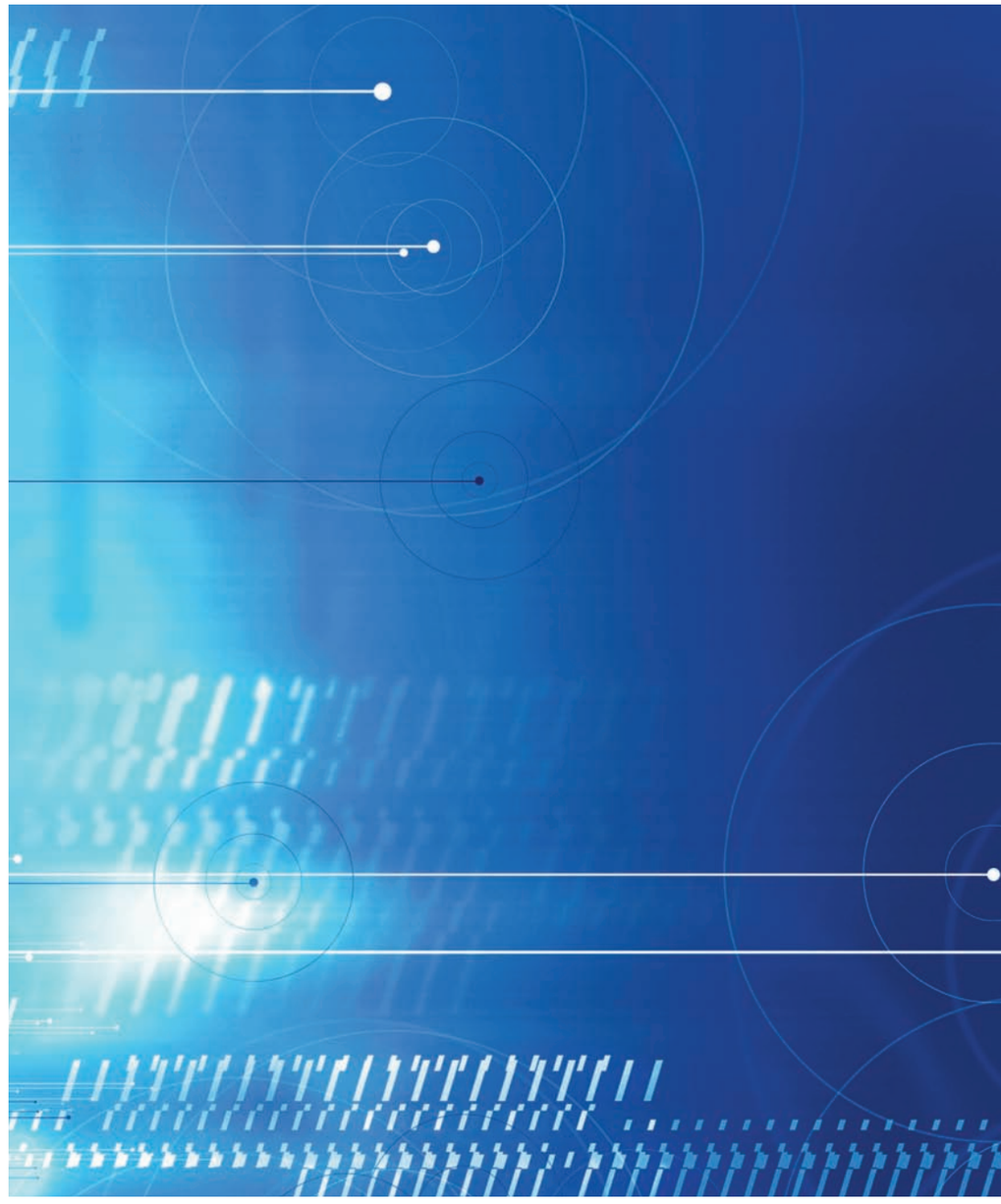
This script will install a few core libraries along with the official Sun (now Oracle) Java 1.6 JDK; what's more, this script also installs Git. At this point, you've got an EC2 instance running in the cloud ready for deployment of a Java application!

EC2, like the rest of AWS, is a game changer. The ability to rapidly provision a wide range of computing resources on a pay-as-you-go basis has ushered in a new era of innovation. Whether you're tinkering in a garage or working for a large company, you have a low-cost option to put working software into production. And should that application become the next Twitter or Facebook, it can quickly scale it to meet user needs.

In a future article, I'll get into the topic of running your own apps on the AWS instances.

— Andrew Glover is the CTO of App47, a company specializing in enterprise mobility. He also is the author of easyb, a BDD framework that won the Jolt Award in 2009. Previously, he was the CEO of Stelligent.

Comment



IN THIS ISSUE

Editorial >>
Cloud Databases >>
AWS >>
Single Inheritance >>
Letters >>
Links >>
Table of Contents >>

From the Vault

Single Inheritance Classes in C

Back in 2006, Ron Kreymborg developed a mechanism for achieving C with Classes without resorting to C++

— *DDJ*

By Ron Kreymborg

I **write a lot of programs for embedded systems**, and I prefer to program in C++ whenever possible. However, for some smaller processors, a C++ compiler either doesn't exist or else is expensive. Recently, I needed to port an existing C++ project to a processor for which I had only a C compiler. The project had benefited greatly from using an inheritable class structure, so I decided to try and develop a mechanism for building inheritable classes in C.

I am not the first to do this, of course, and others have published successful methods. Miro Samek, for instance, developed a set of macros that hide the C code and allow a C++-like syntax (see *Practical State Charts in C/C++*, by Miro Samek, CMP Books, 2002, http://is.gd/TuD67y). Hekon Hallingstad's class implementation used the `offsetof()` macro to access parent class data. Dan Saks described using C abstract types as classes (see "Abstract Types Using C," by Dan Saks, *Embedded Systems Programming*, October 2003). However, I had developed my own standalone class software over the years and my recent work was able to extend this to support classes with single inheritance.

For want of a better name, I use the name of the header file—C_Classes. Where would you use inheritance in embedded systems? Think of manufacturers building a range of chemical process controllers. They all have the same liquid crystal display, analog interfaces, power supply, RS-232 and radio interface, and similar sets of buttons and switches. The software interface to this hardware is also going to be similar. In this article, I introduce inheritance by defining a set of base classes that have drivers for all this hardware and well defined interfaces. Thus, a range of instruments can use the same base class set and only need a set of derived classes that characterize

the particular functionality and set of interfaces that a new instrument will use. This makes for straightforward instrument development.

A Simple Class Structure

C++ hides the class data structure within itself and provides strict rules for how it can be accessed. You don't have this luxury in C, but you can create something similar by defining a single structure to contain all the data belonging to the class.

For example, I use my C_Classes method to define a skeleton class that could be the basis for a motor controller. The class is `Motor` and it controls only the speed and direction. The microprocessor I am programming has a number of ports that are connected to a number of motor controller chips. To set a motor's speed, you need only write the direction and revolutions-per-minute (RPM) to its respective ports. To keep things simple, I just use integers.

The class data structure is defined in the `Motor` class header as a typedef:

```
typedef struct
{
  int*       mPort;
  int        mSpeed;
  DIRECTION  mDirection;
} MotorData;
```

The `MotorData` structure is the definition template for the Motor class data. In C++, these variables would be declared in a private section of the class definition. The mPort class data variable is assigned the port address for a particular motor. This lets the same Motor class code manage several motors in the one application.

The DIRECTION type is a simple enum:

```
typedef enum
{
  DR_UNKNOWN,
  DR_CCW,     // counter-clockwise
  DR_CW,      // clockwise
} DIRECTION;
```

As in C++, the class has a constructor and a destructor. Creating an instance of the class is a call to the constructor using the statement:

```
MotorData* dMotor = MotorCtor();
```

The constructor creates and initializes an instance of the `MotorData` structure just defined, then returns a pointer to it that is assigned to the `dMotor` class data pointer. Subsequently, the caller passes this pointer to the class at every method call. This is the technique the C++ compiler uses, only it does it under the covers. The class contains no instance data within itself: It just provides methods to manipulate the passed-in data.

Okay, that looks simple enough, but how do you call a class method?

Because I am developing a way to create inheritable classes, I need to be able to call methods in a base class through a derived class, add additional methods in that derived class, and override base class methods in the derived class if required. To do this, I build a table of function pointers in RAM that provides access to the public methods of the class. I create the definition of this table of vectors (or `vtable`) as another typedef that includes function pointer prototypes for each public method. In the `Motor` example, you need to be able to initialize the port address, and set/get the speed and direction, all via the class interface, so that makes five methods; see Example 1.

Along with the `vtable` definition, the class header also includes the definition for a pointer to the `vtable` and prototypes for the constructor and destructor. These are all publicly accessible:

```
extern MOTOR* Motor;
extern MotorData* MotorCtor(void);
extern void MotorXtor(MotorData*);
```

Part of the constructor's job is to build in RAM the `vtable` containing the five function pointers and set the public class `vtable` pointer (Motor) to point to that location. After a call to the constructor, users of this class can call any of the public methods through the class pointer. For example, setting the motor speed to 450 RPM, and remembering you also need to pass in the class data pointer, would be the call:

```
Motor->SetSpeed(dMotor,450);
```

While the owner of the `dMotor` structure has no business ever looking inside it, after this call you would know that `dMotor->mSpeed` has a value equivalent to 450 RPM.

The Constructor

So how does the class constructor build the `vtable` and create the data instance?

Again, the header file contains a template of the `vtable` (the `MOTOR` typedef). Within the class code file, an instance of this `vtable` is defined and this instance is copied from ROM to RAM. Continuing the example, in the `Motor` code file, the `vtable` instance along with the global class pointer declaration looks like Example 2(a). It is crucial, of course, that the instance has the same order as the template. The constructor function looks like Example 2(b).

If there is a possibility that the class will have several instances, or be created and destroyed several times, then the class must ensure there is always only one `vtable` in RAM. To this end, every class has `InstCount` as a private variable that is used to keep track of the number of instances extant for this class. It is incremented during construction and decremented during destruction. The actual copying is conveniently done by the `CREATE_VTABLE` macro. This uses `cMotor` as a source, `Motor` as a destination, and the `MOTOR` structure size for both heap allocation and for the number of bytes to copy.

```
typedef struct
{
    //---------------------------------------
    // Set a copy of the port address to use.
    void (*InitPort)(MotorData*, int*);

    //---------------------------------------
    // Set/Get the motor speed.
    void (*SetSpeed)(MotorData*, int);
    int (*GetSpeed)(MotorData*);

    //---------------------------------------
    // Set/Get the motor rotation direction.
    void (*SetDirection)(MotorData*, DIRECTION);
    DIRECTION (*GetDirection)(MotorData*);
} MOTOR;
```

Example 1: Initializing port address, and setting/getting special directions.

As an aside, my original C_Classes version for standalone classes used the simple `Motor = &cMotor` instead of the macro. You can still do this for standalone classes if RAM is in short supply.

Again, the constructor must also create and initialize the class data on the heap for every instantiation. To this end, the constructor creates a pointer that is returned to each individual caller to manage during the life of the class instance. The `Allocate` function is defined in the C_Classes header and is a simple wrapper around malloc that includes an internal allocation counter. The counter can be interrogated at any time and can be useful during debugging.

That's it for the class constructor: It must make sure the `vtable` is set up in RAM and initializes the class data. However, as you'll see, a derived class requires a little more work.

IN THIS ISSUE

Editorial >>
Cloud Databases >>
AWS >>
Single Inheritance >>
Letters >>
Links >>
Table of Contents >>

```
(a)
static const MOTOR cMotor = {
  InitPort,
  SetSpeed,
  GetSpeed,
  SetDirection,
  GetDirection
};
MOTOR* Motor;


(b)
MotorData* MotorCtor(void)
{
    MotorData* d;

    // Create the data structure
    // and initialize the contents.
    d = Allocate(sizeof(MotorData));
    d->mPort = NULL;
    d->mSpeed = 0;
    d->mDirection = DR_UNKNOWN;

    // Create the vtable by
    // copying it to RAM.
    if (InstCount++ == 0)
    {
        CREATE_VTABLE(Motor, cMotor, MOTOR);
    }
    return d;
}
```

Example 2: (a) vtable instance and global class pointer declaration; (b) constructor function.

The Destructor

The code for a normal destructor is straightforward. If the data pointer is valid, the motor is forced to a stop and the actual disposal code is just two lines; see Example 3.

The class data is destroyed when the destructor calls the `Free` function. This is defined in the C_Classes header, and like the `Allocate` function, is a wrapper around the C library malloc/free heap memory

```
void MotorXtor(MotorData* d)
{
  if (d)
  {
      *d->mPort = 0;
        // ensure motor stopped
      Free(d);
      DestroyVtable(Motor, &InstCount);
    }
}
```

Example 3: Destructor.

management. `Free` decrements the internal counter that was incremented by `Allocate`.

The `vtable` class's destruction is handled by the `DestroyVtable` function that is also defined in the C_Classes header. While the constructor increments the class `InstCount` variable, it is decremented in `DestroyVtable`. The counter going to zero implies that this is the last instance of the class to be destroyed, and consequently, the `vtable` is deleted from RAM by a call to `Free`.

All functions in a class code file are declared as static—the only access is via the class pointer. Listings One and Two show the C_Classes header and code files. The `Motor` class header (plain.h) and code files (plain.c) are available for download at http://is.gd/miHw83.

Listing One

```
/*****************************************************************************
   PROJECT  C Classes
   NAME     C_Classes.h
   PURPOSE  A header file for C classes derived using the C_Class methods.
   AUTHOR   Ron Kreymborg, Jennaron Research
*****************************************************************************/
#ifndef _C_CLASSES_H
#define _C_CLASSES_H

#include <stdlib.h>
#include <memory.h>

#define TRUE       1
#define FALSE      0
//---------------------------------------------------------------------------
```

IN THIS ISSUE

Editorial >>
Cloud Databases >>
AWS >>
Single Inheritance >>
Letters >>
Links >>
Table of Contents >>

```
// A macro to copy the vtable structure in rom into ram. The VECTOR parameter
// is the destination structure pointer, and the STRUCTURE parameter is a
// pointer to the source vector table. Note that sizeof will ensure there is
// space for all functions defined in the typedef.
//
#define CREATE_VTABLE(VECTOR, STRUCTURE, TYPEDEF) \
        VECTOR = (TYPEDEF*)Allocate(sizeof(TYPEDEF)); \
        memcpy(VECTOR, &STRUCTURE, sizeof(TYPEDEF))
/*
If you want to replace the macro with a function, here is the
required prototype...
void CREATE_VTABLE(void** to, void* from, int size);
*/
//---------------------------------------------------------------------------
// Free the vtable heap memory based on usage. Both parameters are pointers.
//
extern void DestroyVtable(void*, int*);
//---------------------------------------------------------------------------
// Allocate space from the heap.
//
extern void* Allocate(int);
//---------------------------------------------------------------------------
// Return the memory to the heap.
//
extern void Free(void*);
//---------------------------------------------------------------------------
// Return the current count of allocated items on the heap. Call when the
// process completes to check if all classes have been correctly shutdown.
//
extern int DataCount(void);

#endif  // _C_CLASSES_H
```

Listing Two

```
/*****************************************************************************
   PROJECT  C Classes
   NAME     C_Classes.c
   PURPOSE  The C_Classes vtable heap allocation and de-allocation methods.
   AUTHOR   Ron Kreymborg, Jennaron Research
*****************************************************************************/
#include <stdio.h>
#include "C_Classes.h"

static int mDataCount;
//---------------------------------------------------------------------------
// Destroy the vtable based on whether the instance count goes to zero.
//
void DestroyVtable(void* structure, int* count)
{
    if (--(*count) == 0)
    {
        Free(structure);
    }
}
//---------------------------------------------------------------------------
// Remove a heap structure.
//
void Free(void *p)
{
    mDataCount--;
    free(p);
}
//---------------------------------------------------------------------------
// Allocate the requested bytes from the heap.
//
void *Allocate(int size)
{
    void *p = malloc(size);
    if (p)
    {
        mDataCount++;
        return p;
    }
    else
    {
        // The heap space is exhausted. Provide an error handler here. Do NOT
        // ever return from here unless the heap has been enlarged. Put a
        // suitable error handler here that matches the processor environment
        // requirements.
        //
        printf("Out of heap memory");
        while(1) {;}
    }
}
//---------------------------------------------------------------------------
// Return the number of outstanding heap allocations.
//
int DataCount(void)
{
    return mDataCount;
```

The vtable Copy Macro

I try not to use macros very often, but for the `vtable`, copiers save typing a lot of fiddly and error-prone syntax. The definition in the C_Classes header is:

IN THIS ISSUE

Editorial >>
Cloud Databases >>
AWS >>
Single Inheritance >>
Letters >>
Links >>
Table of Contents >>

```
#define  CREATE_VTABLE(VECTOR,
                       STRUCTURE, TYPEDEF) \
     VECTOR = (TYPEDEF*)Allocate
                       (sizeof(TYPEDEF)); \
              memcpy(VECTOR, &STRUCTURE,
                       sizeof(TYPEDEF))
```

It allocates space on the heap for the structure, assigning it to the `VECTOR` pointer, then copies the class `vtable` instance `STRUCTURE` from ROM to this space in the heap.

(a)

```
typedef struct
{
  MotorData  mMotor;
  int        mAccelRate;
} MotorControlData;
```

(b)

```
typedef struct
{
    //-----------------------------------------------
    // Set the new speed and an accel/decel rate to reach
    // that speed.
    void (*SetNewSpeed)(MotorControlData*, int speed, int rate);

    //-----------------------------------------------
    // Base class Motor methods extracted from its typedef.
    void (*InitPort)( MotorData*, int*);
    void (*SetSpeed)(MotorData*, int);
    int (*GetSpeed)(MotorData*);
    void (*SetDirection)(MotorData*, DIRECTION);
    DIRECTION (*GetDirection)(MotorData*);
} MOTOR_CONTROL;

extern MotorControlData* MotorControlCtor(void);
extern void MotorControlXtor(MotorControlData*);
extern MOTOR_CONTROL* MotorControl;
```

Example 4: (a) Class data structure; (b) typedef for the derived class vtable, and constructor/destructor and class vtable pointer definitions.

Data Hiding

At this point, C++ programmers are grumbling that I have exposed the class's data by publishing the `MotorData` structure in the header. This is true. There is nothing to stop users of this class from typing a statement like:

```
newSpeed = MotorData->speed;
```

rather than calling the appropriate method, and so blowing away the very idea of objects. For example, what if the `Motor` class provided a revolutions-per-minute interface but internally used radians-per-second? To derive from a base class, it must somehow publish its data structure. However, an ordinary class need not. You simply move the data definition into the code file and replace all exposed references to the data structure with void pointers. From the examples so far, the `SetSpeed` definition in the header would then look like:

```
void (*SetSpeed)(void*, int);
```

and the example constructor, method call, and destructor would look like:

```
void* MotorData = MotorCtor();
Motor->SetSpeed(dMotor, 450);
MotorXtor(dMotor);
```

Externally, the data pointer is now useless, but within the code file, a simple macro is used to cast the void pointer. The source-code listings (available electronically) all include the ME macro for reference. Its use in the aforementioned `SetSpeed` method would be:

```
void SetSpeed(void* d,int speed)
{
  ME(d)->mSpeed = speed;
}
```

IN THIS ISSUE

Editorial >>
Cloud Databases >>
AWS >>
Single Inheritance >>
Letters >>
Links >>
Table of Contents >>

Of course, a little source-code sticky-nosing would allow any programmer to learn the class data structure, but I am assuming that those using these methods would not willfully bypass their intent, and using void pointers makes accidental use very unlikely. The downside is you lose the structure cross-checking capability of the compiler.

An Inheritable Class Structure

Inheritance in a C++ context means the ability to both derive a new class from a base class that adds new capability, and perhaps to change some functional aspect of the base class. While C++ has a syntax for this, you must do it yourself in C. The intent of my implementation is to minimize RAM usage and execution time, the former is usually a scarce resource in small microprocessors.

The header file for a derived class includes the same typedef for the class methods as described earlier, but in addition, it now also includes the methods from the base class that are to be available to users of the derived class. I stay with the motor example and derive a class that provides some additional capability, like setting an acceleration rate.

You first need to define the data structure for the derived class. This must be usable by both the derived and base classes. The mechanism I use is based on the fact that, if the base class data structure is the first entry in the derived class data structure, pointers to both a derived and a base class will point at the same data. While this limits the technique to just deriving from one base class (single inheritance), it is rare to find a requirement for multiple inheritance in small embedded systems.

Our derived class data needs to store the acceleration rate as well, so the class data structure becomes Example 4(a). The typedef for the derived class `vtable`, along with the constructor/destructor and class

```
MotorData* MotorCtor(void* data,
                     void(**v)(void)
{
  MotorData* d;
  if (data)
  {
      // This is the base class for a
      // defined class. The caller has
      // defined both their own
      // data structure and vtable.
      d = data;
      if (v)
      {
          *v++ = InitPort;
          *v++ = SetSpeed;
          *v++ = GetSpeed;
          *v++ = SetDirection;
          *v++ = GetDirection;
      }
  }
  else
  {
      // This is a standalone
      // class, and both the
      // data structure and
      // vtable must be created here.
      d = Allocate(sizeof(MotorData));
      if (InstCount++ == 0)
      {
        CREATE_VTABLE(Motor,
                      cMotor, MOTOR);
      }
  }

  // Initialize the class data.
  d->mPort = NULL;
  d->mSpeed = 0;
  d->mDirection = DR_UNKNOWN;

  return d;
}
```

Example 5: Revised Motor constructor.

```
MotorControlData* MotorControlCtor(void)
{
    MotorControlData* d;

    d = Allocate(sizeof(MotorControlData));
    d->mAccelRate = 0;

    if (InstCount++ == 0)
    {
        CREATE_VTABLE(MotorControl, cMotorControl, MOTOR_CONTROL);
        MotorCtor(d, &MotorControl->InitPort);
    }
    else
    {
        MotorCtor(d, NULL);
    }

    return d;
}
```

Example 6: Derived class constructor.

`vtable` pointer definitions, looks like Example 4(b).

The `MotorControl` class `vtable` pointer provides access to this class in the same way the `Motor` pointer did for the `Motor` class. We can use the same technique as before to initialize this pointer, but how do we provide both access to the base class methods and have it initialize our data structure?

Like the `Motor` class, the `MotorControl` class defines an instance of the `vtable` in its code file:

```
static const MOTOR_CONTROL cMotorControl = {
  SetNewSpeed
};
MOTOR_CONTROL* MotorControl;
```

But wait! Where are the base class vectors listed in the typedef?

Step back and think about what is required in a derived class. You don't want the ROM overhead of same name functions in the derived class that just pass execution on to the base class. What you do want is for the derived class to be able to call the base class and have it initialize both the derived class's `vtable` and its base class data, and from then on just provide its methods for use until destructor time.

Harking back to the `Motor` class constructor, I called a macro to construct a copy of the `vtable` instance in RAM. If I use the same macro for the derived class, what happens?

```
CREATE_VTABLE(MotorControl,
        cMotorControl, MOTOR_CONTROL);
```

Because it uses the size of the `MOTOR_CONTROL` structure, it copies the `SetNewSpeed` vector and leaves empty space for the other five vectors. If you now pass to the base class a pointer to where the first base class method should be in the `MotorControl` `vtable`, it can simply copy the addresses of these methods one after another back into the derived class's `vtable`. In the `MotorControl` example, the first base class method is `InitPort`, and if you take its address and pass it to the base class, the mechanism is rather simple. In the derived class:

```
void (**v)() = &(MotorControl->InitPort);
```

and in the base class:

```
*v++ = InitPort;
*v++ = SetSpeed;
etc.
```

You also need the base class to initialize the base class data in the derived class. These requirements mean that, for the base class to initialize the derived class, you must pass to the base class constructor a pointer to the data structure in the derived class and a pointer to the first of the base class vectors in the derived class RAM `vtable`. Example 5, a revised `Motor` constructor to do just that, fills out the derived class RAM `vtable` with pointers to the base class methods.

The derived class data pointer is assigned to the local pointer and subsequently the constructor initializes the data back in the derived class. On the other hand, passing in a couple of nulls sets up a standalone class. (This demonstration constructor is configured to build both derived classes and instances of itself; a rare situation in practice, and a true base class would leave out the standalone code.)

If you now call the `MotorControl SetSpeed` method, it directly executes that method in the base class.

The derived class can override a base class method by overwriting the associated vector in the `vtable`; for example, if the derived class wanted to override the base class `SetDirection` method. Then, after the `vtable` has been built, it could replace that method with a statement such as:

```
MotorControl->SetDirection = MySetDirection;
```

If access to the base class version is still required, you can declare a static function pointer for saving the base class pointer prior to the replacement. Example 6 is the example derived class constructor.

Because in C_Classes the base class for a derived class contains nothing other than the code, the same base class can support a number of derived classes in the one application. In the example I present here, as well as this derived class that provides controlled acceleration for one motor, another derived class could provide pulse width control for a motor attached to a peristaltic pump, with the control input parameter being liters-per-hour.

Motor.h and motor.c are the header and code files (available here) for the now modified `Motor` base class, while MotorControl.h and MotorControl.c show the new `MotorControl` class. MotorVtable.h is the `Motor` base class `vtable` definition in a form that can be included in both the base class itself and any derived classes. Using this technique means there are only two places the programmer must ensure correspondence—the class definition and its instance in the class code file (although it does mean the compiler gives incompatible types warnings). Main.c (available electronically) is a demonstration main program showing examples of using these classes. The ports in the microprocessor are simulated as global integers. It is instructive to create a project in something like Visual C and step through the entire program.

Compiler Dialects

The example listings I've presented are for Microsoft's VC7 compiler, mainly because the Visual C/C++ package is fairly common and has an excellent debugger. Other compilers may require small changes. For example, the GCC compiler versions typically provide the memcpy prototype in the string.h header file rather than the memory.h file.

Most compilers will give you incompatible type warnings when referencing parental structures with derived class names. While these can be safely ignored, they can also be simply fixed with a cast to the base class structure; for example:

```
MotorXtor((MotorData*)d);
```

IN THIS ISSUE

Editorial >>
Cloud Databases >>
AWS >>
Single Inheritance >>
Letters >>
Links >>
Table of Contents >>

I use the standard library memcpy in the macros. On some microprocessors, particularly those with a Harvard architecture, the compiler may require special directions when declaring the class `vtable` instance and copying it from ROM to RAM. Check this out as, otherwise, you will find the compiler copies the instance to RAM before your program runs as part of data initialization, and then memcpy simply copies it from one place in RAM to another.

Conclusion

I have presented a fairly simple method for creating classes in C. The code is transparent and regular, in that the method can apply to both standalone and inheritable classes with no change in syntax and little additional code. An inherited class need only call the base class constructor to be fully initialized. The ROM footprint is hardly different from a carefully modularized traditional C method, with the advantage of an obvious class structure.

While not a problem in itself, the fact that the class `vtable` definition and its instance are in two different places means that you must take care they always match. This is particularly so when adding new functions to an existing design.

In a typical small embedded system, classes are created early and rarely destroyed, in part to ensure the heap does not become fragmented over the months or years between resets, but also because in most cases all classes are in use all the time. If your app is dynamically creating and destroying classes, you need to be sure of your compiler's heap management. My recommendation is to get all heap setup over and done with during initialization.

— At the time this article was originally published, Ron was designing software and hardware for embedded systems.

Comment

IN THIS ISSUE

Editorial >>
Cloud Databases >>
AWS >>
Single Inheritance >>
Letters >>
Links >>
Table of Contents >>

This Month on DrDobbs.com

Interesting items posted on www.drdobbs.com over the past month that you may have missed

WHAT'S NEW IN SUBVERSION 1.7

A completely redesigned working copy library speeds common operations, streamlined communication between clients and servers, and much more.

http://drdobbs.com/open-source/231500282

JAVA SE 7: NEW FILE IO

Java SE 7 has many enhancements and new APIs.. Eric Bruno talks briefly about the new Java.NIO 2.0 APIs, and specifically about how much easier it is to work with files.

http://drdobbs.com/blogs/java/231600403

THE EMERGING JAVASCRIPT REVOLUTION

JavaScript's ubiquity, speed, and easy syntax are suddenly making it the default scripting language for a wide range of apps.

http://drdobbs.com/open-source/231600203

MALLOC MADNESS

If you get a good return from `malloc`, does that ensure you actually allocated memory?

http://drdobbs.com/blogs/embedded-systems/231600776

RUNNING CUDA CODE NATIVELY ON X86 PROCESSORS

A new compiler from PGI makes it possible to use the same CUDA code on x86 processors, Nvidia chips, or both; so programmers now have the ability to use a single source tree of CUDA code to reach those customers who own CUDA-enabled GPUs or who use x86-based systems.

http://drdobbs.com/high-performance-computing/231500166

USER-DEFINED HASH FUNCTIONS FOR UNORDERED MAPS IN C++0X

The basic hash map should be the first- or second-most used container in your arsenal, but unless you're a C++ savant, you might find yourself ditching it out of sheer frustration.

http://drdobbs.com/windows/231600210

JVM LANGUAGE SUMMIT 2011 VIDEOS

The JVM Language Summit is an annual event sponsored by Oracle, attended by implementers of JVM-based languages and programming tools. Slides for each presentation accompany the videos and are available for download on the individual pages.

http://drdobbs.com/java/231002486

IN THIS ISSUE

Editorial >>
Cloud Databases >>
AWS >>
Single Inheritance >>
Letters >>
Links >>
Table of Contents >>

Dr. Dobb's

Andrew Binstock Editor in Chief, Dr. Dobb's
alb@drdobbs.com

Deirdre Blake Managing Editor, Dr. Dobb's
dblake@techweb.com

Amy Stephens Copyeditor, Dr. Dobb's
astephens@techweb.com

Sean Coady Webmaster, Dr. Dobb's
scoady@techweb.com

Jon Erickson Editor in Chief Emeritus , Dr. Dobb's

DR DOBB'S
UBM TECHWEB
303 Second Street,
Suite 900, South Tower
San Francisco, CA 94107
1-415-947-6000

IN THIS ISSUE

Editorial >>
Cloud Databases >>
AWS >>
Single Inheritance >>
Letters >>
Links >>
Table of Contents >>

Dr.Dobb's Business Contacts

DR. DOBB'S

Sales Director, Michele Hurabiell
(415) 378-3540, mhurabiell@techweb.com

Account Executive, Shaina Guttman
(212) 600-3106, sguttman@techweb.com

INFORMATIONWEEK BUSINESS TECHNOLOGY NETWORK

CMO, Scott Vaughan
(949) 223-3662, svaughan@techweb.com

VP of Group Sales, InformationWeek Business Technology Network, Martha Schwartz
(212) 600-3015, mschwartz@techweb.com

Sales Assistant, Group Sales, Kelly Glass
(212) 600-3327, kglass@techweb.com

Publisher's Assistant, Esther Rodriguez
(949) 223-3656, erodriguez@techweb.com

SALES CONTACTS—WEST

Western U.S. (Pacific and Mountain states) and Western Canada (British Columbia, Alberta)

Inside Sales Manager, Vesna Beso
(415) 947-6104, vbeso@techweb.com

Sales Assistant, Ian Doyle
(415) 947-6105, idoyle@techweb.com

Strategic Accounts

Account Director, Sandra Kupiec
(415) 947-6922, skupiec@techweb.com

Account Manager, Shoshana Freisinger
(415) 947-6349, sfreisinger@techweb.com

Sales Assistant, Matthew Cohen-Meyer
(415) 947-6214, mmeyer@techweb.com

SALES CONTACTS—EAST

Midwest, South, Northeast U.S. and Eastern Canada (Saskatchewan, Ontario, Quebec, New Brunswick)

District Manager, Jenny Hanna
(516) 562-5116, jhanna@techweb.com

District Manager, Michael Greenhut
(516) 562-5044, mgreenhut@techweb.com

Account Manager, Cori Gordon
(516) 562-5181, cgordon@techweb.com

Inside Sales Manager East, Ray Capitelli
(212) 600-3045, rcapitelli@techweb.com

Sales Assistant, Elyse Cowen
(212) 600-3051, ecowen@techweb.com

Strategic Accounts

District Manager, Mary Hyland
(516) 562-5120, mhyland@techweb.com

Account Manager, Tara Bradeen
(212) 600-3387, tbradeen@techweb.com

Account Manager, Jennifer Gambino
(516) 562-5651, jgambino@techweb.com

Sales Assistant, Kathleen Jurina
(212) 600-3170, kjurina@techweb.com

Dr. Dobb's

Sales Director, Michele Hurabiell
(415) 378-3540, mhurabiell@techweb.com

Account Executive, Shaina Guttman
(212) 600 3106, sguttman@techweb.com

MARKETING

VP, Marketing, Winnie Ng-Schuchman
(631) 406-6507, wng@techweb.com

Marketing Manager, Monique Luttrell
(949) 223-3609, mluttrell@techweb.com

AUDIENCE DEVELOPMENT

Director, Karen McAleer
(516) 562-7833, kmcaleer@techweb.com

BUSINESS OFFICE

General Manager, Marian Dujmovits

United Business Media LLC
600 Community Drive
Manhasset, N.Y. 11030 (516) 562-5000

UBM TECHWEB

Tony L. Uphoff CEO

John Dennehy CFO

David Michael CIO

Bob Evans Sr. VP and Global CIO Director

Joseph Braue Sr. VP, Light Reading Communications Network

Scott Vaughan CMO

Ed Grossman Executive Vice President, InformationWeek Business Technology Network

John Ecke VP and Group Publisher, Financial Technology Network, InformationWeek Government, InformationWeek Healthcare

Martha Schwartz VP, Group Sales, InformationWeek Business Technology Network

Beth Rivera Senior VP, Human Resources

David Berlind Chief Content Officer, TechWeb, and Editor in Chief, TechWeb.com

Fritz Nelson VP, Editorial Director, InformationWeek Business Technology Network, and Executive Producer, TechWeb TV

UNITED BUSINESS MEDIA LLC

Pat Nohilly Sr. VP, Strategic Development and Business Admin.

Marie Myers Sr. VP, Manufacturing